Tzong-Chen Wu
Chin-Laung Lei
Vincent Rijmen
Der-Tsai Lee (Eds.)

LNCS 5222

Information Security

11th International Conference, ISC 2008
Taipei, Taiwan, September 2008
Proceedings

Springer

Lecture Notes in Computer Science 5222

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Tzong-Chen Wu Chin-Laung Lei
Vincent Rijmen Der-Tsai Lee (Eds.)

Information Security

11th International Conference, ISC 2008
Taipei, Taiwan, September 15-18, 2008
Proceedings

Springer

Volume Editors

Tzong-Chen Wu
National Taiwan University of Science and Technology
Department of Information Management
No. 43, Sec. 4, Keelung Road, Taipei 106, Taiwan
E-mail: tcwu@cs.ntust.edu.tw

Chin-Laung Lei
National Taiwan University, Department of Electrical Engineering
No. 1, Sec. 4, Roosevelt Road, Taipei 106, Taiwan
E-mail: lei@cc.ee.ntu.edu.tw

Vincent Rijmen
Graz University of Technology
Institute for Applied Information Processing and Communications (Austria)
Katholieke Universiteit Leuven, Department of Electrical Engineering (Belgium)
Inffeldgasse 16a, 8010 Graz, Austria
E-mail: Vincent.Rijmen@iaik.tugraz.at

Der-Tsai Lee
Academia Sinica, Institute of Information Science
No. 128, Sec. 2, Academia Road, Nankang, Taipei 115, Taiwan
E-mail: dtlee@iis.sinica.edu.tw

Library of Congress Control Number: 2008933846

CR Subject Classification (1998): E.3, E.4, D.4.6, K.6.5, C.2

LNCS Sublibrary: SL 4 – Security and Cryptology

ISSN 0302-9743
ISBN-10 3-540-85884-9 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-85884-3 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12517565 06/3180 5 4 3 2 1 0

Preface

The 11th Information Security Conference (ISC 2008) was held in Taipei, Taiwan, September 15–18, 2008. ISC is an annual international conference covering research in theory and applications of information security. It was first initiated as a workshop (ISW) in Japan in 1997. This was followed by ISW 1999 in Malaysia and ISW 2000 in Australia. ISW became ISC when it was held in Spain in 2001 (ISC 2001). The latest conferences were held in Brazil (ISC 2002), UK (ISC 2003), USA (ISC 2004), Singapore (ISC 2005), Greece (ISC 2006), and Chile (ISC 2007). This year the event was sponsored by the Chinese Cryptology and Information Security Association (Taiwan), the Taiwan Information Security Center of the Research Center for IT Innovation (Academia Sinica, Taiwan), the National Taiwan University of Science and Technology (Taiwan), the NTU Center for Information and Electronics Technologies (Taiwan), Academia Sinica (Taiwan), the National Science Council (Taiwan), the Ministry of Education (Taiwan), the Taipei Chapter of the IEEE Computer Society (Taiwan), BankPro E-service Technology Co., Ltd. (Taiwan), Exsior Data & Information Technology, Inc. (Taiwan), Giga-Byte Education Foundation (Taiwan), Hewlett-Packard Taiwan, Hivocal Technologies, Co., Ltd. (Taiwan), Microsoft Taiwan, Paysecure Technology Co., Ltd. (Taiwan), Symlink (Taiwan), and Yahoo! Taiwan Holdings Limited (Taiwan Branch).

In order to cover the conference's broad scope, this year's main Program Committee consisted of 61 experts from 22 countries. Additionally, the conference also featured a special AES Subcommittee, chaired by Vincent Rijmen (Graz University of Technology, Austria).

The conference received 134 submissions from 31 countries, 33 (including 4 in the AES special session) of which were selected by the committee members for presentation at the conference, based on quality, originality and relevance. Each paper was anonymously reviewed by at least three committee members. In order to encourage and promote student participation, the ISC 2008 Program Committee selected three student-coauthored papers for the Best Student Paper award, one from each region: Asia, Europe, and the Americas. The papers were, respectively, "Deterministic Constructions of 21-Step Collisions for the SHA-2 Hash Family," by Somitra Sanadhya and Palash Sarkar (Indian Statistical Institute, India), "Collisions for RC4-Hash," by Sebastiaan Indesteege and Bart Preneel (Katholieke Universiteit Leuven, Belgium), and "Proxy Re-signatures in the Standard Model," by Sherman S.M. Chow (New York University, USA) and Raphael Phan (Loughborough University, UK). The program also included invited speeches by Doug Tygar (UC Berkeley, USA) and Tatsuaki Okamoto (NTT, Japan).

Many people helped to make ISC 2008 successful. We would like to thank all those who contributed to the technical program and to organizing the conference. We are very grateful to the Program Committee members and the external referees for their efforts in reviewing and selecting the papers. We would like to express our

special thanks to all the organizing committee members for making the conference possible. We also give our thanks to all the authors of the submitted papers and the invited speakers for their contributions to the conference.

July 2008

Tzong-Chen Wu
Chin-Laung Lei

ISC 2008

The 11th International Security Conference
Taipei, Taiwan, September 15–18, 2008

General Chair

Der-Tsai Lee	Director of the Institute of Information Science, Academia Sinica, Taiwan

Steering Committee

Ed Dawson	Queensland University of Technology, Australia
Der-Tsai Lee	Academia Sinica, Taiwan
Javier López	University of Málaga, Spain
Masahiro Mambo	University of Tsukuba, Japan
Eiji Okamoto	University of Tsukuba, Japan
René Peralta	NIST, USA
Rebecca Wright	Rutgers University, USA
Yuliang Zheng	University of North Carolina at Charlotte, USA

Program Co-chairs

Tzong-Chen Wu	National Taiwan University of Science and Technology, Taiwan
Chin-Laung Lei	National Taiwan University, Taiwan

Finance Chairs

Jason Shih	Paysecure Technology Co., Ltd., Taiwan
Tim Chiou	Microsoft Corporation
David Fan	Exsior Data & Information Technology, Inc., Taiwan

Publications Chairs

D.J. Guan	National Sun Yat-Sen University, Taiwan
Wen-Guey Tzeng	National Chiao Tung University, Taiwan
Chun-I Fan	National Sun Yat-Sen University, Taiwan

Local Arrangements Chairs

Hahn-Ming Lee	National Taiwan University of Science and Technology, Taiwan
Ren-Junn Hwang	Tamkang University, Taiwan
Ming-Hour Yang	Chung Yuan Christian University, Taiwan

Registration Chairs

Wei-Hua He	Soochow University, Taiwan
Shiuh-Jeng Wang	National Central Police University, Taiwan
Wen-Shenq Juang	National Kaohsiung First University of Science and Technology, Taiwan

Publicity Chairs

Chung-Huang Yang	National Kaohsiung Normal University, Taiwan
Hung-Min Sun	National Tsing Hua University, Taiwan
Chien-Lung Hsu	Chang Gung University, Taiwan

Web Masters

Bo-Yin Yang	Institute of Information Science, Academia Sinica, Taiwan
Chun-Yang Chen	Institute of Information Science, Academia Sinica, Taiwan
Chen-Mou Cheng	National Taiwan University, Taiwan

Program Committee

Mikhail Atallah	Purdue University, USA
Feng Bao	Institute for Infocomm Research, Singapore
David Basin	ETH Zurich, Switzerland
Josh Benaloh	Microsoft Research, USA
Alex Biryukov	University of Luxembourg, Luxembourg
Johannes Buchmann	TU Darmstadt, Germany
David Chadwick	University of Kent, UK
Tsuhan Chen	Carnegie Mellon University, USA
Tzi-Cker Chiueh	State University of New York at Stony Brook, USA
Debbie Cook	Bell Labs, USA
Robert Deng	Singapore Management University, Singapore
Xiaotie Deng	City University of Hong Kong, China
Claudia Diaz	Katholieke Universiteit Leuven, Belgium
Jintai Ding	University of Cincinnati, USA
Chun-I. Fan	National Sun Yat-Sen University, Taiwan
Pierre-Alain Fouque	ENS, France
Juan Garay	Bell Labs, USA
Shai Halevi	IBM Research, USA
Wei-Hua He	Soochow University, Taiwan
Amir Herzberg	Bar-Ilan University, Israel

Dennis Hofheinz	CWI, Netherlands
Lei Hu	State Key Laboratory of Information Security, China
Ren-Junn Hwang	Tamkang University, Taiwan
Marc Joye	Thomson R&D, France
Wen-Shenq Juang	National Kaohsiung First University of Science and Technology, Taiwan
Hiroaki Kikuchi	Tokai University, Japan
Kwangjo Kim	Information and Communication University, Korea
Seungjoo Kim	Sungkyunkwan University, Korea
Marcos Kiwi	University of Chile, Chile
Spyros Kokolakis	University of the Aegean, Greece
Steve Kremer	ENS Cachan, France
Xuejia Lai	Shanghai Jiao Tong University, China
Ruby Lee	Princeton University, USA
San Ling	Nanyang Technological University, Singapore
Subhamoy Maitra	Indian Statistical Institute, India
Keith Martin	RH University of London, UK
Fabio Massacci	University of Trento, Italy
Breno de Medeiros	Google, USA
Chris Mitchell	RH University of London, UK
Atsuko Miyaji	JAIST, Japan
Fabian Monrose	Johns Hopkins University, USA
Hikaru Morita	Kanagawa University, Japan
David Naccache	Gemplus, France
Koji Nakao	KDDI, Japan
Kaisa Nyberg	Helsinki University of Technology and Nokia, Finland
Carles Padró	Polytechnic University of Catalonia, Spain
Adrian Perrig	Carnegie Mellon University, USA
Andreas Pfitzmann	Dresden University of Technology , Germany
Raphael Phan	Loughborough University, UK
Josef Pieprzyk	Macquarie University, Australia
Rei Safavi-Naini	University of Calgary, Canada
Kouichi Sakurai	Kyushu University, Japan
Pierangela Samarati	University of Milan, Italy
Angelos Stavrou	George Mason University, USA
Dominique Unruh	Saarland University, Germany
Ariel Waissbein	ITBA and Core Security, Argentina
Felix Wu	UC Davis, USA
Huaxiong Wang	Nanyang Technological University, Singapore
Bo-Yin Yang	Academia Sinica, Taiwan
Kan Yasuda	NTT, Japan
Heung Youl Youm	Soonchunhyang University/IITA, Korea

AES Subcommittee

Joan Daemen	STMicroelectronics Belgium, Belgium
Xuejia Lai	Shanghai Jiao Tong University, China
Chi Sung Laih	National Cheng Kung University, Taiwan
Vincent Rijmen	Graz University of Technology, Austria
Matt Robshaw	France Telecom, France
Hung-Min Sun	National Tsing Hua University, Taiwan
Ralph Wernsdorf	Rohde & Schwarz, Germany

External Reviewers

Guido Bertoni	Stijn Lievens
Rainer Böhme	Chu-Hsing Lin
Elie Bursztein	Hubert Comon-Lundh
Kostas Chatzikokolakis	Serdar Pehlivanoglu
Jung-Hui Chiu	Duong Hieu Phan
Kim-Kwang Raymond Choo	Natalya Rassadko
Sherman S.M. Chow	Ermaliza Razali
Ricardo Corin	Ayda Saidane
Oriol Farras	Stefan Schiffner
Hani Hassen	Sandra Steinbrecher
Matt Henricksen	Ruggero Susella
Alejandro Hevia	Carmela Troncoso
Vladimir Kolesnikov	Guilin Wang
Gabriel Kuper	Shiuh-Jeng Wang
Cedric Lauradoux	Artsiom Yautsiukhin
Jia-Hong Lee	Sung-Ming Yen

Sponsoring Institutions

Chinese Cryptology and Information Security Association (CCISA), Taiwan
Taiwan Information Security Center (TWISC), Center for IT Innovation, Academia Sinica, Taiwan
National Taiwan University of Science and Technology (NTUST), Taiwan
NTU Center for Information and Electronics Technologies (NTU CIET), Taiwan
Academia Sinica, Taiwan
National Science Council (NSC), Taiwan
Ministry of Education, Taiwan
IEEE Computer Society, Taipei Chapter
BankPro E-service Technology Co., Ltd. (Taiwan)
Exsior Data & Information Technology, Inc. (Taiwan)

Giga-Byte Education Foundation (Taiwan)
Hivocal Technologies, Co., Ltd. (Taiwan)
Microsoft Taiwan
Paysecure Technology Co., Ltd. (Taiwan)
Symlink (Taiwan)
Yahoo! Taiwan Holdings Limited (Taiwan Branch)

Table of Contents

Trusted Computing

Database and System Security

Intrusion Detection

Network Security

Cryptanalysis

Digital Signatures

AES Special Session

Symmetric Cryptography and Hash Functions

Authentication

Security Protocols

Property-Based TPM Virtualization

Ahmad-Reza Sadeghi[1], Christian Stüble[2], and Marcel Winandy[1]

[1] Ruhr-University Bochum, D-44780 Bochum, Germany
{ahmad.sadeghi,marcel.winandy}@trust.rub.de
[2] Sirrix AG security technologies, Lise-Meitner-Allee 4
D-44801 Bochum, Germany
stueble@sirrix.com

Abstract. Today, virtualization technologies and hypervisors celebrate their rediscovery. Especially migration of virtual machines (VMs) between hardware platforms provides a useful and cost-effective means to manage complex IT infrastructures. A challenge in this context is the virtualization of hardware security modules like the Trusted Platform Module (TPM) since the intended purpose of TPMs is to securely link software and the underlying hardware. Existing solutions for TPM virtualization, however, have various shortcomings that hinder the deployment to a wide range of useful scenarios. In this paper, we address these shortcomings by presenting a flexible and privacy-preserving design of a virtual TPM that in contrast to existing solutions supports different approaches for measuring the platform's state and for key generation, and uses property-based attestation mechanisms to support software updates and VM migration. Our solution improves the maintainability and applicability of hypervisors supporting hardware security modules like TPM.

1 Introduction

Corporate computing today is characterized by enterprises managing their own IT infrastructure. In his article, "The end of corporate computing" [1], Nicholas G. Carr predicts a shift from holding corporate assets to purchasing services from third parties. Similar to electricity suppliers, there would be enterprises offering IT functionality to other companies. *Virtualization* technology would be one of the key drivers of the changing IT paradigm.

Indeed, virtualization enables the deployment of standardized operating environments on various hardware platforms, features the execution of several virtual machines (VMs) on a single platform, and allows to suspend a VM and resume it at a later time. An important feature of virtualization is that one can migrate a VM between hardware platforms, which allows an easy transfer of working environments, e.g., in case of hardware replacements or switching to another computer. Moreover, Virtual Machine Monitors (VMM), or *hypervisors*, are also known to be an efficient way to increase the security of computer systems [2]. They provide isolation between VMs by mediating access to hardware resources

T.-C. Wu et al. (Eds.): ISC 2008, LNCS 5222, pp. 1–16, 2008.

and by controlling a rather simple interface of resources compared to a full operating system. Thus, different environments can be protected against harm from other environments or violations of user privacy. For instance, an employee can simply separate home and office usage in VMs.

Trusted Computing is considered to be another promising concept to improve trustworthiness and security of computer systems. The Trusted Computing Group (TCG), an industrial initiative towards the realization of Trusted Computing, has specified security extensions for commodity computing platforms. The core TCG specification is the *Trusted Platform Module* (TPM) [3,4], currently implemented as cost-effective, tamper-evident hardware security module embedded in computer mainboards. The TPM provides a unique identity, cryptographic functions (e.g., key generation, hash function SHA-1, asymmetric encryption and signature), protected storage for small data (e.g., keys), and monotonic counters (storing values that can never decrease). Moreover, it provides the facility to securely record and report the platform state (so-called integrity measurements) to a remote party. The platform state typically consists of the hardware configuration and the running software stack, which is measured (using cryptographic hashing) and stored in the TPM. Several operating system extensions [5,6] already support the TPM as underlying security module.

In this context, the combination of virtualization and trusted computing provides us with new security guarantees such as assurance about the booted VMM, but it also faces us with new challenges. On the one hand, VMs should be flexible to support migration. On the other hand, security modules like the TPM act as the *root of trust* attached to the hardware, and must be shared by various VMs. Hence, different approaches for TPM virtualization have already been proposed [7,8,9]. Being able to migrate a VM together with its associated virtual TPM (vTPM) is of special importance to guarantee the availability of protected data and cryptographic keys after migration. However, the existing solutions have some shortcomings which strongly limit their deployment: After migrating a VM and its vTPM to another platform with different integrity measurements than the source platform, or after performing an authorized update of software, the VM cannot access cryptographic keys and the data protected by those keys anymore. This hinders the flexibility of migrating the VM to a platform providing the same security properties but different integrity measurements as the source platform. Moreover, differentiated strategies for key generation and usage are missing. Some IT environments demand for cryptographic keys generated and protected by the hardware TPM while some VMs would benefit from the performance of software keys. In addition, some VMs can be migratable while others must not be.

Contribution. In this paper we address these problems in the following way:

- We propose a vTPM architecture that supports various functions to measure the state of the platform, various usage strategies for cryptographic keys, and both based on a user-defined policy of the hypervisor system (Sect. 5).
- We show how the new measurement functions of our vTPM can be used to realize property-based attestation and sealing. Our design can protect user

privacy by filtering properties to be attested in order to not disclose the particular system configuration (Sect. 6).
– We allow a transparent migration of vTPM instances to platforms with a different binary implementation and show this is possible without losing the strong association of security properties (Sect. 7).

Moreover, our design does not require to modify the software of a VM (except for the driver in the guest OS that interfaces to the vTPM instead of the hardware TPM). Existing TPM-enabled applications directly profit from the flexibility of the underlying vTPM. We expect furthermore that our vTPM design can be realized based on other secure coprocessors [10,11] because of its flexibility and high-level abstraction of functionality.

Outline. We describe typical use cases that need flexible vTPMs in Sect. 2 and define corresponding requirements in Sect. 3. Section 4 considers background of the TPM and discusses the related work. We present our contribution in Sect. 5, 6, and 7, whereas we address in Sect. 8 how we achieve the requirements.

2 Use Case Scenario: Corporate Computing

We consider the use case in a corporate setting as our running example to make various essential requirements on VMs and vTPMs more clear. Nevertheless, these requirements also hold for many other applications such as e-government, grid computing, and data centers.

Suppose an enterprise employee uses a laptop for both corporate and private tasks which run in isolated VMs (Fig. 1).

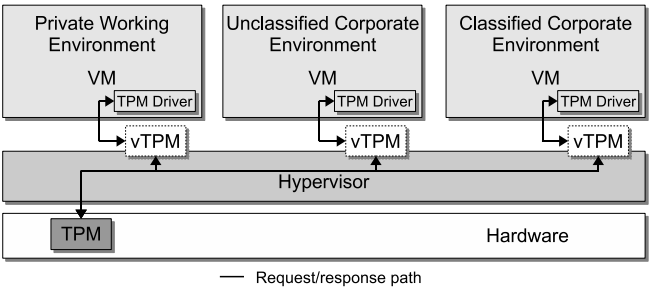


Fig. 1. Private and corporate working environments with virtual TPMs

Private working environment: This environment may use the TPM, e.g., to protect the key of a hard-disk encryption program or the reference values of an integrity checker. Using existing vTPM approaches, the protected data would become unavailable if the user updates a software component within the VM.

Unclassified corporate environment is for processing unclassified data of the company. Users should be able to migrate this VM to their computer at

home to continue working. After migration, access to protected data and reporting integrity measurements of the VM should still be possible as long as the underlying platform conforms to the company's security policy.

Classified corporate environment: This environment is for processing classified data. Hence, it has stronger security requirements regarding the usage of encryption keys. To access a corporate VPN, the company's security policy may require this environment to be bound to this specific hardware and that the cryptographic keys are protected by a physical TPM.

3 Requirements on TPM Virtualization

The scenarios described above show the need for a flexible vTPM architecture that supports all required functionalities. We consider the main requirements of such an architecture below, where we add new requirements *R5-R8* to those (*R1-R4*) already identified by [7].

R1 Confidentiality and integrity of vTPM state: All internal data of a vTPM (keys, measurement values, counters, etc.) have to be protected against unauthorized access.

R2 Secure link to chain of trust: There must be an unforgeable linkage between the hardware TPM and each vTPM as well as between the VM and its associated vTPM. This includes trust establishment by managing certificate chains from the hardware TPM to vTPMs.

R3 Distinguishability: Remote parties should be able to distinguish between a real TPM and a vTPM since a virtual TPM may have different security properties than a physical one.

R4 Uncloneability and secure migration: The state of a vTPM shall be protected from cloning, and it can be securely (preserving integrity, confidentiality, authenticity) transferred to another platform if the destination platform conforms to the desired security policy.

R5 Freshness: The vTPM state shall not be vulnerable to replay attacks (e.g., an adversary shall not be able to reset the monotonic counters of a vTPM).

R6 Data availability: Data sealed by a vTPM should be accessible if the platform provides the desired security properties. This should also hold after migration or software updates.

R7 Privacy: Users should be able to decide which information about the platform state (configuration of hardware and hypervisor) is revealed to a VM or to a remote party.

R8 Flexible key types: Different protection levels and implementations of cryptographic keys should be supported (as described in the use case scenarios).

As we will discuss later, the existing vTPM solutions do not fulfill all requirements of the typical use cases as described in Sect. 2.

4 Background and Related Work

4.1 The Trusted Platform Module

The TPM has two main key (pairs): the *Endorsement Key* (EK) representing the TPM's identity and the *Storage Root Key* (SRK), used to encrypt other keys generated by the TPM (which are stored outside the TPM). The TPM supports *trusted boot* by allowing to record measurements of the hardware configuration and software stack during the boot process. These measurements (typically, SHA-1 hash of binaries) are stored in specific TPM registers called *Platform Configuration Registers* (PCRs). Adding a hash m to a PCR is called extension and requires to use the function TPM_Extend(i, m), which concatenates m to the current value of the i-th PCR by computing a cumulative hash.

Based on these PCR values, the TPM provides the *sealing* functionality, i.e., binding encrypted data to the recorded configuration, and *attestation*, i.e., reporting the system state to a (remote) party. The latter uses the function TPM_Quote, which presents the recorded PCR values signed by an *Attestation Identity Key* (AIK) of the TPM. The AIK plays the role of a pseudonym of the TPM's identity EK for privacy reasons, but to be authentic the AIK must be certified by a trusted third party called Privacy-CA.

4.2 Integrity Measurement

AEGIS [12] performs an integrity check during the boot process of the operating system and builds a chain of trust based on root reference values protected by special hardware. Enforcer [13] is a Linux kernel security module operating as integrity checker for file systems. It uses a TPM to verify the integrity of the boot process and to protect the secret key of an encrypted file system. IMA [6] inserts measurement hooks in functions relevant for loading executable code in Linux in order to extend the measurement chain to the application level.

Enforcer and IMA are examples of TPM-enabled applications which could be used and executed in a VM that has a vTPM.

4.3 Property-Based Attestation

TCG binary attestation has some important drawbacks: (i) disclosure of platform configuration information could be abused for platform tracking (*privacy*) and discriminating against specific system configurations; (ii) lack of flexibility, i.e., data bound to a particular configuration is rendered inaccessible after system migration, update or misconfiguration (*data availability*); (iii) less scalability due to necessary management of every trusted platform configuration. To tackle these problems, property-based approaches were proposed in the literature (see below): Instead of attesting hash values of binaries, they attest abstract properties describing the behavior of a program or system, e.g., that the hypervisor is certified according to a certain Common Criteria protection profile. The advantage is that properties can remain the same even if the binaries change.

Haldar et al. [14] present an approach exploiting security properties of programming languages, e.g., type-safety. This allows to provide a mechanism for runtime attestation. However, it requires a trusted language-specific execution environment and is limited to applications written in that language.

Jiang et al. [15] have shown that it is possible to have certificates stating that the keyholder of a certain public key has a desired property, e.g., to be an application running inside an untampered secure coprocessor.

A pragmatic approach for property-based attestation uses property certificates [16,17,18]. A trusted third party (TTP) issues certificates $cert(pk_{TTP}, p, m)$, signed by the TTP's public key pk_{TTP}, and stating that a binary with hash m has the property p. When a PCR of the TPM is going to be extended with a measurement value, a *translation* function looks for a matching certificate. If the function can find and verify a matching certificate, it extends the PCR with the public key pk_{TTP} or, as proposed by [19], with a bit string representation of p. If no certificate is found or the verification fails, the PCR is extended with zero.

While these approaches can be applied to existing TPMs or vTPMs by adding the translation function to a trusted component outside of the (v)TPM, we apply the translation functions inside our vTPM (Sect. 5.1)[1]. This allows us to control the translation in each vTPM instance individually and reduces the dependency of external software components (e.g., running in VMs).

4.4 Trusted Channel

A *trusted channel* is a secure channel with the additional feature that it is bound to the configuration of the endpoint(s). An attestation (binary or property-based) of the involved endpoint(s) is embedded in the establishment of the secure channel [20,21]. Hence, each endpoint can get an assurance whether the counterpart complies with trust requirements before the channel is settled. Asokan et al. [22] describe a protocol which creates a secret encryption key that is bound not only to the TPM of the destination platform, but also to the configuration of the trusted computing base (TCB). Binding a key to the configuration of the underlying TCB has been used with TPM [13] and secure coprocessors [15,10].

4.5 TPM Virtualization

Berger et al. [7] propose an architecture where all vTPM instances are executed in one special VM. This VM provides a management service to create vTPM instances and to multiplex the requests. To protect the vTPM state when it is stored on persistent memory, the state is encrypted using the sealing function of the physical TPM. Optionally, the vTPM instances may be realized in a secure coprocessor card. Compared to a real TPM, the vTPM has a different certificate for its vEK, e.g., including a statement that it is virtual. Thus, a verifying party will be able to distinguish between a vTPM and a TPM. The authors

[1] We use the simple version of property certificates, e.g., issued by a corporate CA, certifying "approved by IT department".

discuss different strategies for trust establishment, i.e., the way new certificates are issued for a vTPM: (a) The vEK is signed by the AIK of the physical TPM and the vTPM requests certificates for its vAIKs at a privacy CA. (b) The TPM directly signs the vAIK with its AIK. (c) A local CA issues a certificate for the vEK of the vTPM.[2] In order to extend the chain of trust, they link the vTPM to its underlying TCB by mapping the lower PCRs of the real TPM to the lower PCRs of a vTPM. This is supposed to enable the vTPM to include the configuration of the underlying hypervisor platform during attestation.

However, this approach has the restriction that after migrating a VM and its vTPM to a different hypervisor platform, the VM cannot access data that was sealed by the vTPM on the source platform (*R6 data availability*). In our approach, we show how property-based measurement can be realized in the vTPM while the interface to the VM remains the same as for binary attestation. This removes the restriction that migration is only possible between binary identical platforms. Moreover, our design allows flexible key types (*R8*) and protects privacy (*R7*) by allowing to filter the information to be revealed during attestation.

GVTPM [9] is an architectural framework that supports various TPM models and different security profiles for each VM under the Xen hypervisor [23]. The authors discuss two different vTPM models: software-based and hardware-based. The former generates and uses cryptographic keys entirely in software, whereas the latter uses the keys of the physical TPM. GVTPM is not limited to TPM functionality and may be generalized to any security coprocessor. This is similar to our approach since we also use a high-level abstraction of TPM functionality. However, they realize flexible key types with different vTPM models, whereas our vTPM design can support both. Moreover, GVTPM does not address our requirements of data availability (*R6*) and privacy (*R7*).

Anderson et al. [24] realize the implementation of vTPM instances as isolated domains instead of running all vTPMs in one privileged VM. Except for the implementation, they provide no new aspects of the vTPM, but refer to [7]. Our architecture can also execute vTPM instances in isolated domains since our approach does not depend on a specific implementation.

Goldman and Berger [8] have specified additional commands that would be needed to enhance a physical TPM to directly support VMs. The realization is similar to [7], except that the vTPM-specific functions are realized within the hardware TPM. Hence, they do not address data availability (*R6*) and privacy (*R7*). Moreover, there is no such enhanced TPM chip model available at present.

5 Flexible vTPM Architecture

This section describes the general design of our vTPM architecture. For each VM that needs a vTPM, there is a separate vTPM instance. We assume the underlying hypervisor to protect the internal state and operations of each vTPM from any unauthorized access. This can be achieved by using a secure hypervisor

[2] For our example scenario, we can choose the certificate strategy (c) since the employee's company could serve as a local CA to issue these certificates.

as proposed in [25,26], which enforces access control to resources and controls communication between virtual machines. A VM can only access its associated vTPM via the vTPMInterface.

Figure 2 shows the logical design of our vTPM. The main building blocks are the following: PropertyManagement represents the virtual PCRs and manages different mechanisms to store and read measurement values (Sect. 5.1); KeyManagement is responsible for creating and loading keys (Sect. 5.2); vTPMPolicy holds the user-defined policy of the vTPM instance (Sect. 5.3); CryptographicFunctions provide monotonic counters, random number generation, hashing, etc.; MigrationController is responsible for migrating the vTPM to another platform.

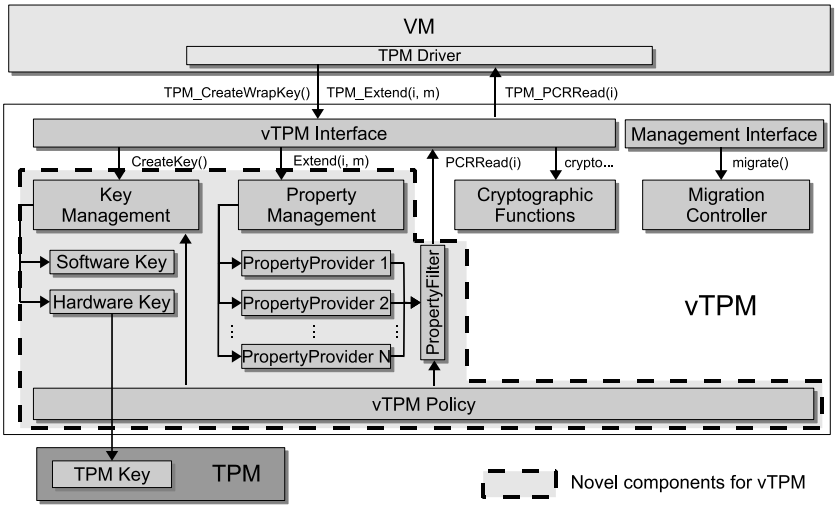


Fig. 2. Logical architecture of the vTPM

5.1 Property Management and Property Providers

To improve flexible migration and to preserve the availability of sealed data after migration or software updates, an essential step is to support other measurement strategies. Applying property-based measurement and attestation [15,19,17,18] to a vTPM allows much more flexibility in the choice of the hypervisor and for easier updates of applications — a VM can still use sealed data or run attestation procedures if the properties of the programs remain the same (see Sect. 4.3).

We define the process of recording measurements into the TPM in a more general way. Therefore, we redefine the extension function of the TPM:

$$\mathsf{Extend}(i,\ m)\text{: } PCR_i \leftarrow \mathsf{translate}(PCR_i, m).$$

In case of the TCG specification, translate is $SHA1(PCR_i||m)$.

Our vTPM design is based on a plug-in-like architecture for various vPCR extension strategies. Each extension strategy is realized by a PropertyProvider

module implementing another translate() function. To add measurement values to the PCRs of the vTPM (vPCRs), the guest OS in a VM simply uses the standard TPM_Extend() function, specifying the PCR number i and the hash data m to be stored. The PropertyManagement calls each property provider to extend the corresponding vPCR with the measured data value. Each PropertyProvider applies its translation function on the data and stores the resulting value in the corresponding vPCR field. The general form of the PCR extension is as follows:

$$\mathsf{PropertyProvider}_j.\mathsf{Extend}(i,\ m)\text{: } vPCR_{i,j} \leftarrow \mathsf{translate}_j(vPCR_{i,j}, m)$$

Note that each PropertyProvider has its own vector of virtual PCRs. Thus there is a matrix of vPCR values for each vTPM, as depicted in Fig. 3. The way how to store the vPCR values is up to the implementation of each property provider. One could cumulatively hash all input values, as the TCG version of Extend. An alternative is to simply concatenate the inputs on each invocation of Extend.

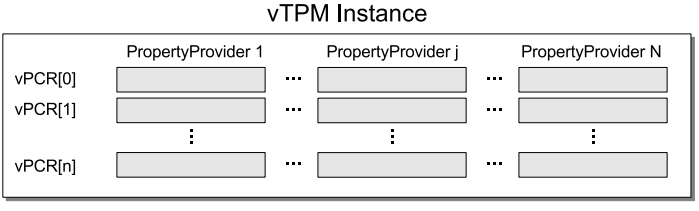


Fig. 3. Matrix of vPCRs for a vTPM instance

To give an example of different property providers, consider the virtual machine VM_k wants to extend PCR_i with a hash value m of a binary, e.g., when the guest OS within VM_k loads and measures a software component. The vTPM instance $vTPM_k$ is associated with VM_k. Suppose there are two PCR extension strategies, a HashProvider and a CertificateProvider. The HashProvider extends PCR_i with the hash m as provided by the VM. The CertificateProvider, however, looks for a property certificate (see Sect. 4.3).

In this example, the vTPM actually has two PCRs for PCR_i, i.e., $vPCR_{i.hash}$ and $vPCR_{i.cert}$. However, when VM_k requests to read the current PCR value, e.g., by invoking the function TPM_PCRRead(i), the VM is only aware of an abstract PCR_i and the returned data must be of fixed-length for compliance to the TCG specification. This is achieved by the PropertyFilter that defines, based on vTPMPolicy, which property provider has to be used when reading this particular vPCR. The responsible provider then returns the requested value.

5.2 Flexible Key Generation and Usage

To achieve a flexible key usage, the KeyManagement hides details of different strategies to create cryptographic keys when a VM requests a new key pair. The keys can be generated as software keys in the vTPM and as a result they are

protected as part of the vTPM's state. Alternatively, the vTPM can delegate the key generation to a physical security module, e.g., a TPM or a smartcard. In this case, the keys are protected by the hardware.

For example, in our "classified" corporate VM scenario, it is required to have an encryption key protected by the physical TPM. When the VM requests to create the key at the vTPM, the KeyManagement delegates the request directly to the hardware TPM. Note that the VM cannot decide which key type to be used; instead, this is decided by the vTPM policy.

Although the vTPMPolicy can specify which type of key is to be used, not all combinations are possible. A vTPM cannot use a hardware AIK to sign the vPCRs because the vTPM does not possess the private key part of the AIK. However, the realization of KeyManagement is not limited to software and physical TPMs. Instead, the underlying flexibility allows the realization based on different hardware security modules while providing VMs compatibility to the TCG specification.

5.3 User-Defined vTPM Policy

The user of the hypervisor system can specify a vTPMPolicy per vTPM instance when the instance is created. The policy specifies what information about the system state is actually visible to the VM and, hence, to other systems the VM is allowed to communicate with. This is possible due to the selection of property providers, which define possible translations of measurement values. For all vTPM operations, the policy defines what property provider has to be used. For example, a policy can define to always use the CertificateProvider for sealing operations requested by the VM in order to enable flexible migration to a certified platform.

For each vTPM instance, the vTPMPolicy specifies the key strategy to be used. In this way, we can source out privacy issues the VM would have to handle otherwise. For instance, the policy decides when to use a particular vAIK and how often it can be used until the KeyManagement has to generate a new one.

5.4 Initialization of the vTPM

On its instantiation, the vTPM creates a new Endorsement Key (vEK) and a new Storage Root Key (vSRK). Certificates for the vEK and for vAIKs can be issued, e.g., by a local CA.

Existing vTPM solutions [7] propose to directly map the lower PCRs of the physical TPM to the lower vPCRs of the vTPM. These PCRs contain measurements of the BIOS, the bootloader, and the hypervisor. While this provides a linkage to the underlying platform, it is based on the hash values of binary code only, which hinders migration as discussed earlier.

In our solution, we map these PCR values by applying our property providers and build up a vPCR matrix, holding a vector of vPCRs for each property provider. How the mapping is actually done is up to the implementation of the property providers. After initialization of the platform by means of trusted

boot, the physical TPM contains the measurements of the platform configuration. When a new vTPM instance is created by the hypervisor, the PropertyManagement of this vTPM requests the physical TPM to read out all PCRs, i.e., from PCR_0 to PCR_n. Then each property provider is invoked with the following function:

$$\mathsf{PropertyProvider}_j.\mathsf{initVirtualPCRs}\ (PCR_0,...,PCR_n)$$

For example, $\mathsf{PropertyProvider}_A$ could map the values of PCR_0,...,PCR_7 one to one to $vPCR_{0.A}$,...,$vPCR_{7.A}$, whereas $\mathsf{PropertyProvider}_B$ could accumulate somehow all physical measurements into one single vPCR. Finally, $\mathsf{PropertyProvider}_C$ could translate the PCR values into properties using certificates. This approach allows to support different mapping strategies simultaneously.

By defining the vTPM policy accordingly, we can control which mapping will be used later. For instance, to support availability of sealed data after migration, we can define to use the certificate-based property provider when the VM wants to seal data to $vPCR_0$,...,$vPCR_7$. If flexible migration should not be allowed, we would define to use $\mathsf{PropertyProvider}_A$, resulting in sealing data to binary measurements of the underlying platform.

6 Realizing Property-Based Functionality with vTPM

In this section we describe how we can use the feature of property providers to realize property-based attestation and property-based sealing in the vTPM.

6.1 Property-Based Attestation

The CertificateProvider is one example of a property provider that uses property certificates issued by a TTP. As mentioned in Sect. 5.1, CertificateProvider applies its translation function to extend $vPCR_{i.cert}$ with the TTP's public key pk_{TTP}. The attestation protocol works as follows: A verifier requests attestation to (PCR_i,...,PCR_j) of VM_k; the VM requests its vTPM to *quote* the corresponding vPCRs with the key identified by $vAIK_{ID}$:

$$(pcrData,\ sig) = vTPM_k.\mathsf{Quote}(vAIK_{ID},\ nonce,\ [i,...,j])$$

where *pcrData* denotes the quoted vPCR values, *sig* denotes the vTPM's signature on *pcrData* and *nonce*. Internally, the PropertyManagement of the vTPM decides according to the vTPMPolicy which property provider is to be used for attestation. If the CertificateProvider is chosen, then $vTPM_k$ will use its vAIK as identified by $vAIK_{ID}$ to sign the values of $vPCR_{[i,...,j].cert}$.

The verifier verifies the signature *sig* and whether *pcrData* represent the desired properties. Hence, we can use vTPMPolicy to restrict attestation to certain property providers, depending on the use case. This allows to control which information about the VM and the user's system is going to be revealed to a remote party and as a result fulfills our privacy requirement.

6.2 Property-Based Sealing

The sealing procedure of our vTPM works as follows. A virtual machine VM_k chooses a handle *vBindkeyID* of a binding key that was previously created in the virtual TPM instance $vTPM_k$, and then issues the sealing command to seal *data* under the set of virtual PCRs (PCR_i,...,PCR_j). The vTPM realizes the sealing function as follows:

$vTPM_k$.Seal(*vBindkeyID*, [*i*,...,*j*], *data*):
provider := vTPMPolicy.askForProvider([*i*,...,*j*]);
FOR $l := i$ TO j DO $prop_l :=$ provider.PCRRead(l);
$pk :=$ KeyManagement.getPublicKey(*vBindkeyID*);
$ed :=$ encrypt[pk]($i||prop_i||...||j||prop_j||data$);
return *ed*.

The vTPM asks its vTPMPolicy which property provider to use, which can depend on the combination of vPCRs for the sealing operation. It requests the KeyManagement to load the corresponding binding key, retrieves the vPCR values of the specified PropertyProvider, and encrypts *data*, and the vPCR values with corresponding vPCR number. When the VM wants to unseal the data again, the vTPM proceeds as follows:

$vTPM_k$.UnSeal(*vBindkeyID*, *ed*):
$(sk, pk) :=$ KeyManagement.getKeyPair(*vBindkeyID*);
$(i||prop_i||...||j||prop_j||data) :=$ decrypt[sk](*ed*);
provider := vTPMPolicy.askForProvider([*i*,...,*j*]);
FOR $l := i$ TO j DO BEGIN
  $prop'_l :=$ provider.PCRRead(l);
  if ($prop'_l \neq prop_l$) return $\emptyset$;
END
return *data*.

The vTPM first loads the binding key pair identified by *vBindkeyID* and decrypts the sealed data *ed*. The vTPMPolicy decides again which PropertyProvider to use. The current vPCR values are compared to the values stored in the sealed data. Only if all matching pairs are equal, the plain *data* is returned to VM_k.

Of course, a property provider like CertificateProvider is needed as one possible way to realize property-based sealing. This is especially interesting if sealing is related to software components in the VM. Depending on the realization of the property provider, unsealing will be possible if the measured applications of the VM are changed but still provide the same properties, i.e., the corresponding property certificate is available and valid.

Moreover, property-based sealing enables the availability of sealed data after migration of a VM and its corresponding vTPM to a platform with a different binary implementation. This can be achieved, e.g., by using a CertificateProvider for the $vPCR_{[0,...,7].cert}$, representing the properties of the underlying hypervisor platform. This measurement does not change after migration to a target platform having a certificate stating the same properties.

7 Migration of vTPM

Our vTPM migration protocol is based on the vTPM migration protocol in [7]. However, in contrast to [7] we do not use a migratable[3] TPM key to protect the session key but rather we propose to embed the migration procedure in a *trusted channel*. As described in Sect. 4.4, the trusted channel allows to create a secret encryption key that is bound not only to the TPM of the destination platform but also to the configuration of its TCB. In our case the TCB comprises the vTPM and the hypervisor. The advantage of using such a trusted channel is that, once it has been established, it can be re-used for migration of several vTPM instances between the same physical platforms. Moreover, a transfer can even securely occur after the target machine has rebooted.

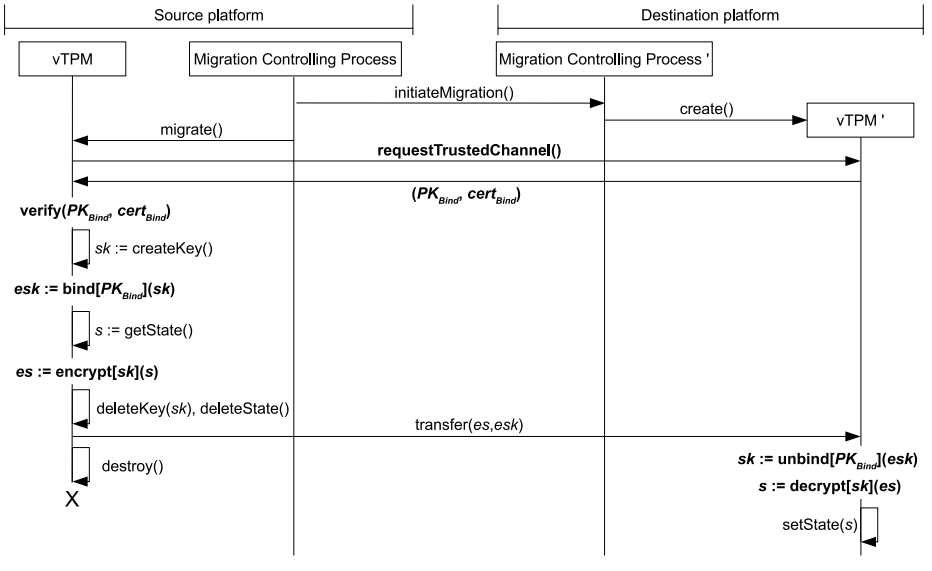


Fig. 4. A vTPM migration based on a trusted channel

Figure 4 shows our migration procedure, based on the trusted channel protocol of [22].The process (of the hypervisor) responsible for migrating the VM also initiates the migration of the associated vTPM. After creating a new vTPM instance on the target system, the source vTPM requests to establish a trusted channel to the destination vTPM. When the trusted channel is successfully established, the source vTPM encrypts its state and transfers it to the destination. The source vTPM destroys itself subsequently, i.e., the vTPM deletes its own state from memory. On the target, the vTPM decrypts and activates the state.

Additionally, there is another issue if the hypervisor supports to suspend a vTPM, i.e., if the vTPM state was stored on persistent memory before. If

[3] There are various attributes for TPM keys. Migratable keys are allowed to be migrated to another TPM.

the suspended vTPM state is sealed to the hardware TPM (see Sect. 4.5), a migration of the suspended vTPM state ("offline migration") is not possible. However, we can resume a suspended vTPM (i.e., unseal the vTPM state) on the source platform, migrate the vTPM state to the target, and suspend and seal the vTPM state on the target platform to its hardware TPM, respectively. To ensure that the vTPM state is unique and cannot be reactivated at the source platform, the hypervisor has also to delete the key used to seal the vTPM state.

In order to prevent data loss from transmission failures during migration, the encrypted vTPM state can be stored persistently before transmission so that the state can be transmitted again to the target platform (if the migration is still pending and the keys of the trusted channel are still valid). Based on the ideas of [11], the encrypted state could be deleted on the source after the source receives an acknowledgment from the target.

8 Requirements Revisited

We briefly address the requirements of Sect. 3. Our architecture supports *flexible key types* by means of KeyManagement (Sect. 5.2). We have addressed *data availability* with PropertyManagement (Sect. 5.1) and property-based sealing (Sect. 6). To protect *privacy*, we make use property-based attestation and PropertyFilter, which controls the disclosure of properties according to the vTPM policy. The *inclusion in the chain of trust* is realized by mapping the PCRs of the physical TPM to the vTPM (Sect. 5.4). The requirement of *distinguishability* was already addressed by prior work (see Sect. 4.5). To protect the *confidentiality and integrity of vTPM state* and to maintain *uncloneability*, we can also resort to existing approaches, which we briefly discuss below.

Runtime protection of the vTPM state is assumed to be provided by the hypervisor through isolation. But to enable a VM and its vTPM to suspend and resume, all data belonging to the state of vTPM instance need to be protected against copying clones to other platforms or replaying old states on the local platform. In case the vTPM state has to be stored on persistent memory, prior work [7] encrypts the vTPM state using a key that is sealed to the state of PCRs in the hardware TPM, i.e., binding it to the configuration of the TCB.

To prevent a local replay of an old vTPM state, the sealed state has to be stored on storage providing freshness. For instance, [22] proposes a solution based on monotonic counters of the TPM. To prevent a replay of migration, the target platform needs to be able to detect the freshness of the transferred vTPM state. In [22] and [7], the source encrypts the data to be transferred together with a unique nonce that was defined by the target platform.

9 Conclusion and Future Work

We have presented a flexible and privacy-preserving design for virtual TPMs that supports different approaches for measuring the platform's state and for key generation. We have demonstrated that our design allows to implement

property-based sealing and attestation in a vTPM. This enables the availability of protected data and cryptographic keys of the vTPM after migrating to another platform that provides the same security properties but may have a different binary implementation. TPM-enabled applications executed in a VM can directly profit from this flexibility without the need for modification.

The vTPM design is part of a security architecture that we currently implement. We are going to decompose the vTPM functionality into several services that can be used as required. Future work also includes the evaluation of performance and scalability. Moreover, flexible offline migration of vTPM states is an open issue which we will work on.

References

1. Carr, N.G.: The end of corporate computing. MIT Sloan Management Review 46(3), 67–73 (2005)
2. Karger, P.A., Zurko, M.E., Bonin, D.W., Mason, A.H., Kahn, C.E.: A VMM security kernel for the VAX architecture. In: Proceedings of the IEEE Symposium on Research in Security and Privacy, pp. 2–19. IEEE Computer Society, Los Alamitos (1990)
3. Trusted Computing Group: TPM Main Specification Version 1.1b (February 2002), https://www.trustedcomputinggroup.org
4. Trusted Computing Group: TPM Main Specification Version 1.2 rev. 103 (July 2007), https://www.trustedcomputinggroup.org
5. Microsoft Corporation: Bitlocker drive encryption (July 2007), http://www.microsoft.com/technet/windowsvista/security/bitlockr.mspx
6. Sailer, R., Zhang, X., Jaeger, T., van Doorn, L.: Design and implementation of a TCG-based integrity measurement architecture. In: 13th Usenix Security Symposium, San Diego, California (August 2004), pp. 223–238 (2004)
7. Berger, S., Caceres, R., Goldman, K.A., Perez, R., Sailer, R., van Doorn, L.: vTPM: Virtualizing the Trusted Platform Module. In: Proceedings of the 15th USENIX Security Symposium, USENIX, August 2006, pp. 305–320 (2006)
8. Goldman, K., Berger, S.: TPM Main Part 3 – IBM Commands (April 2005), http://www.research.ibm.com/secure_systems_department/projects/vtpm/mainP3IBMCommandsrev10.pdf
9. Scarlata, V., Rozas, C., Wiseman, M., Grawrock, D., Vishik, C.: TPM virtualization: Building a general framework. In: Pohlmann, N., Reimer, H. (eds.) Trusted Computing, Vieweg, pp. 43–56 (2007)
10. Smith, S.W., Weingart, S.: Building a high-performance, programmable secure coprocessor. Computer Networks 31(8), 831–860 (1999)
11. Yee, B.S.: Using Secure Coprocessors. PhD thesis, School of Computer Science, Carnegie Mellon University (May 1994) CMU-CS-94-149
12. Arbaugh, W.A., Farber, D.J., Smith, J.M.: A secure and reliable bootstrap architecture. In: Proceedings of the IEEE Symposium on Research in Security and Privacy, Oakland, CA, May 1997, pp. 65–71. IEEE Computer Society Press, Los Alamitos (1997)
13. Macdonald, R., Smith, S., Marchesini, J., Wild, O.: Bear: An open-source virtual secure coprocessor based on TCPA. Technical Report TR2003-471, Department of Computer Science, Dartmouth College (2003)

14. Haldar, V., Chandra, D., Franz, M.: Semantic remote attestation: A virtual machine directed approach to trusted computing. In: USENIX Virtual Machine Research and Technology Symposium (2004)
15. Jiang, S., Smith, S., Minami, K.: Securing web servers against insider attack. In: 17th Annual Computer Security Applications Conference (ACSAC) (2001)
16. Chen, L., Landfermann, R., Loehr, H., Rohe, M., Sadeghi, A.R., Stüble, C.: A protocol for property-based attestation. In: STC 2006: Proceedings of the First ACM Workshop on Scalable Trusted Computing, pp. 7–16. ACM Press, New York (2006)
17. Poritz, J., Schunter, M., Van Herreweghen, E., Waidner, M.: Property attestation—scalable and privacy-friendly security assessment of peer computers. Technical Report RZ 3548, IBM Research (May 2004)
18. Sadeghi, A.R., Stüble, C.: Property-based attestation for computing platforms: Caring about properties, not mechanisms. In: The 2004 New Security Paradigms Workshop. ACM Press, New York (2004)
19. Kühn, U., Selhorst, M., Stüble, C.: Realizing property-based attestation and sealing with commonly available hard- and software. In: STC 2007: Proceedings of the 2nd ACM Workshop on Scalable Trusted Computing, pp. 50–57. ACM Press, New York (2007)
20. Goldman, K., Perez, R., Sailer, R.: Linking remote attestation to secure tunnel endpoints. In: STC 2006: Proceedings of the First ACM Workshop on Scalable Trusted Computing, pp. 21–24 (2006)
21. Stumpf, F., Tafreschi, O., Röder, P., Eckert, C.: A robust integrity reporting protocol for remote attestation. In: Proceedings of the Second Workshop on Advances in Trusted Computing (WATC 2006 Fall), Tokyo (December 2006)
22. Asokan, N., Ekberg, J.E., Sadeghi, A.R., Stüble, C., Wolf, M.: Enabling fairer digital rights management with trusted computing. In: Garay, J.A., Lenstra, A.K., Mambo, M., Peralta, R. (eds.) ISC 2007. LNCS, vol. 4779, pp. 53–70. Springer, Heidelberg (2007)
23. Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Pratt, I., Warfield, A., Barham, P., Neugebauer, R.: Xen and the art of virtualization. In: Proceedings of the ACM Symposium on Operating Systems Principles, October 2003, pp. 164–177 (2003)
24. Anderson, M.J., Moffie, M., Dalton, C.I.: Towards trustworthy virtualisation environments: Xen library os security service infrastructure. Technical Report HPL-2007-69, Hewlett-Packard Laboratories (April 2007)
25. Sadeghi, A.R., Stüble, C., Pohlmann, N.: European multilateral secure computing base - open trusted computing for you and me. Datenschutz und Datensicherheit DuD, Verlag Friedrich Vieweg & Sohn, Wiesbaden 28(9), 548–554 (2004)
26. Sailer, R., Valdez, E., Jaeger, T., Perez, R., van Doorn, L., Griffin, J.L., Berger, S.: sHype: Secure hypervisor approach to trusted virtualized systems. Technical Report RC23511, IBM Research Division (February 2005)

A Demonstrative Ad Hoc Attestation System

Endre Bangerter[1], Maksim Djackov[2], and Ahmad-Reza Sadeghi[3]

[1] Bern University of Applied Sciences, Switzerland
endre.bangerter@jdiv.org
[2] Bern University of Applied Sciences, Switzerland
dkm1@bfh.ch
[3] University of Bochum, Germany
ahmad.sadeghi@trust.rub.de

Abstract. Given the growing number and increasingly criminally motivated attacks on computing platforms, the ability to assert the integrity of platform becomes indispensable. The trusted computing community has come up with various remote attestation protocols that allow to assert the integrity of a remote platform over a network.

A related problem is that of ad hoc attestation, where a user walks up to a computing platform and wants to find out whether *that* platform in front of her is in a trustworthy state or not. ad hoc attestation is considered to be an open problem, and some very recent publications have outlined a number of open challenges in this field. Major challenges are (i) the security against platform in the middle attacks (ii) viable choice of the device used to perform attestation, and (iii) the manageability of integrity measurements on that device.

In this paper we describe a concrete implementation of an ad hoc attestation system that resolves these challenges. Most importantly, our system offers a novel and very intuitive user experience. In fact, from a user perspective, ad hoc attestation using our solution roughly consists of initiating the process on the target platform and then holding a security token to the screen of the target platform. The outcome of the ad hoc attestation (i.e., whether the platform is trustworthy or not) is then shown on the token's display. This usage paradigm, which we refer to as *demonstrative ad hoc attestation*, is based on a novel security token technology, which we have used. We believe that our system has the potential to be evolved into a system for real world usage.

Keywords: Trusted computing, attestation, Kiosk computing, platform integrity, smart cards.

1 Introduction

Attacks on computing platforms are growing rapidly, becoming more sophisticated, and are increasingly criminally motivated. Just one example of such recent attacks are transaction generators, which take over correctly authenticated e-banking sessions and perform undiscoverable fraudulent transactions [CJM07, Kre]. The Trusted Computing Group (TCG) aims at providing means towards tackling these problems. One of the mechanisms proposed by the TCG is remote attestation [Tru05, Tru04, Trua]. A *remote*

T.-C. Wu et al. (Eds.): ISC 2008, LNCS 5222, pp. 17–30, 2008.

attestation protocol allows a verifier to learn the integrity state of a remote computing platform (the *target platform*) over a network connection. To this end the target platform is equipped with a Trusted Platform Module (TPM), which measures and reports its integrity status. The design and implementation of such protocols has received attention within the trusted computing community, and many variants of such protocols exist [FSE06, GPS06, RS04, GSS+07].

There is a practically relevant application scenario which is not covered by remote attestation protocols. In that scenario, a user walks up to a computing platform and wants to find out whether *that* platform in front of her, which she can identify physically (e.g., by seeing or touching it) is in a trustworthy state or not. Technically, this scenario boils down to finding out whether the platform in question contains a TPM and, if so, getting an integrity measurement from the TPM residing within the platform. To actually obtain that information the user will have to make use of some sort of portable computing device (e.g., mobile phone, smart card etc.), which runs a protocol with the target platform in question. We call protocols that solve the problem underlying this scenario *ad hoc attestation protocol* and the portable device the *user device*. The term "ad hoc" refers to the fact that the user device and the target platform have, loosely speaking, never met before and hence do not share any kind of a priori information (such as cryptographic keys, certificates, identifiers etc.) on each other.

While remote attestation and ad hoc attestation have the same goal - getting the integrity measurements on some target platform, they differ in how the platform is identified. It is the "ad hoc" and "physical identification by the user" aspects which are not covered by remote attestation that make ad hoc attestation a challenging problem.

There is a large number of practically relevant application scenarios for ad hoc attestation. One class of examples is the verification of the platform integrity before performing critical transactions, such as e-banking, accessing sensible data, issuing a digital signature, editing a confidential document etc. Another class is using an unknown computer, e.g., at a friend's place, in an Internet cafe, in a company branch office, etc. Finally, there is a large field of potential future applications where ad hoc attestation could be used to prevent fraud resulting from attacks against common infrastructure, such as payment terminals [DM07], automated teller machines, etc.

Designing practically usable and secure ad hoc attestation protocols and systems is considered to be an open problem. The pertaining challenges are the subject of very recent publications by McCunem et al [MPSvD07], Garriss et al [SG07], as well as of some earlier work by Ranganathan [Ran04]. The challenges described in these works fall into three main categories:

1. *Security against platform in the middle attacks.* The fundamental challenge is "How to detect whether the user is really receiving an integrity measurement of the platform she has in front of her?". In fact, in a *platform in the middle attack*, a corrupted platform could relay attestation requests to a trustworthy platform and thus impersonate the latter.
2. *Viable choice of user device and usability.* For ad hoc attestation to be viable in practice, the user device being chosen and the ad hoc attestation protocol should fulfill certain criteria: It should be based on affordable commodity hardware and feature a small form factor so that it can easily be carried about. It shall offer

universal connectivity between the user device and the target platform, such that essentially any platform can be ad hoc attested. The user device itself has to be trustworthy and resilient against attacks; otherwise ad hoc attestation protocols can be broken by attacking the user device. Last but not least, the device and the ad hoc attestation protocol shall be intuitive and easy to use.

3. *Evaluation and management of integrity measurements.* Assuming that a user device gets the actual integrity measurements of the platform in question, it still needs to evaluate the measurements to decide whether the platform is in a trustworthy state or not. Therefore it must match the integrity measurements against a database of known trustworthy states. The challenge here is to manage such databases on user devices, which often have only limited storage and computing power.

Another major open challenge is to devise run-time attestation techniques, which overcome the limitations of current file integrity based techniques [ESvD05].

Our contributions. In this paper we describe a secure and easy to use ad hoc attestation system, which - we believe - can be evolved in a system for real world usage. More precisely, we describe protocols and a concrete implementation of an ad hoc attestation system that solve the challenges $1-3$ outlined above; we do not tackle the run-time attestation issue, which remains an open research question.

Our ad hoc attestation system makes essential use of a novel security token technology [AXS]. The token's form factor corresponds essentially to that of a conventional smart card. Yet, it has some distinct features, which play an important role in our system. The token features a display, a fingerprint reader for user authentication and trackpad-like navigation, and an optical sensor for receiving data. The optical sensor is crucial since it allows to receive data from a PC by simply holding the token to a PC's screen, where an animated *flickering pattern* - encoding the data - is displayed (for illustrations see Fig. 2 in §3).

These features of the token allow us to resolve the second class of challenges mentioned above. For instance, thanks to the optical sensors, we achieve an unparalleled connectivity since no cabling between the user device and target platform is required. Moreover, the token features an isolated execution architecture with a minimal firmware, which is amenable to assurance techniques. This is an important precondition for the security of the user device.

Most importantly, our system offers a novel and very intuitive user experience. In fact, from a user perspective, ad hoc attestation using our solution roughly consists of initiating the process on the target platform and then holding the token to the screen of the target platform. The outcome of the attestation protocol (i.e., whether the platform is trustworthy or not) is then shown on the token's display. From a usability perspective, "holding the token to the screen of the platform to be verified" is compelling and a highly intuitive usage metaphor resembling the "demonstrative identification" metaphor proposed for ad hoc authentication [DBW02]. We refer to this usage metaphor by *demonstrative ad hoc attestation*.

To tackle the third of the above challenges, we have chosen a server based ad hoc attestation architecture for our implementation. Thereby, on a high level, attestation is performed by a central server and the attestation outcome is then transmitted to the

user device. Thus, all integrity measurements and other attestation related information is managed by a central server. This model integrates very well with the way IT infrastructure is run nowadays. As an example, in the enterprise setting the attestation server could be operated by the IT department and integrated with existing asset and systems management solutions. An other possible setting is where such servers are run by security services companies, which maintain a database of trustworthy integrity measurements, very much like anti-virus companies or managed security solution providers maintain lists of virus or IDS attack signatures.

Outline. In §2 we describe more precisely what an ad hoc attestation protocol is, as well as desirable security properties of such protocols. Then, in §3 we describe the security token technology underlying our results. Our main results follow in §4, where we describe our implementation of a demonstrative ad hoc attestation system. Finally, in §5 we review related work and then go over to conclusions and future work in §6.

2 Ad Hoc Attestation – Basic Notions and Security Goals

In this section we describe what an ad hoc attestation protocol is, as well as the desired security properties of such protocols.

By an *integrity measurement* (denoted by integrity(P)) we refer to a procedure that runs on a platform P and outputs information about the integrity of P. We assume that a measurement is either `good` (i.e., integrity$(P) \in \mathscr{G}$) or `bad` (i.e., integrity$(P) \in \mathscr{B}$). Informally, integrity$(P) \in \mathscr{G}$ means that P is trustworthy, and integrity$(P) \in \mathscr{B}$ that P is corrupted. The current integrity measurement technique used in Trusted Computing, is roughly to compute the hash-values of the disk images of various files residing on P. This, and other tasks, are performed using a Trusted Platform Module (TPM) chip built into P. A novel area of research is that of run-time attestation, where an integrity measurement additionally includes run-time information on the processes running on P [ESvD05].

Consider a user U equipped with a portable *user device* D (e.g., a PDA, smart phone, smart card). An *ad hoc attestation protocol* is a protocol where U *physically identifies* the *target platform* P (i.e., U can see and physically interact with P), and where D, after interacting with P, and possibly third parties, eventually outputs `good` if integrity$(P) \in \mathscr{G}$ or `bad` when integrity$(P) \in \mathscr{B}$. An important point is that since we are looking at the "ad hoc" scenario, one cannot assume that U or D have a priori information (e.g., an identifier of P, pre shared keys etc.) about P.

We call an ad hoc attestation protocol *secure*, if in the presence of an adversary it holds that: If integrity$(P) \in \mathscr{B}$, then D never[1] outputs `good` at the end of a protocol execution, and if integrity$(P) \in \mathscr{G}$, then D never outputs `bad` at the end of a protocol execution.

An often cited security challenge is that of platform in the middle attacks, where a bad target platform impersonates a good platform.

[1] Actually, using the term "never" is too strong. An adversary has always at least a small probability of breaking a system, e.g., by guessing crypto keys etc. A formally satisfactory approach would be to replace never with negligible probabilities - as it is common in cryptography and complexity theory. Since this paper is practically minded we refrain from this formalism.

3 Overview of Token Technology

Our ad hoc attestation system described in §4 makes essential use of unique features of the Axsionics security token system [AXS], which consists of *security tokens* and of a *token server*. In the standard usage scenario of the system, the token allows users to securely authenticate and confirm transactions (e.g., e-banking payments), even in the setting where the user's PC is controlled by an attacker (e.g., malware [CJM07, Kre]). The system can also be used for conventional user authentication.

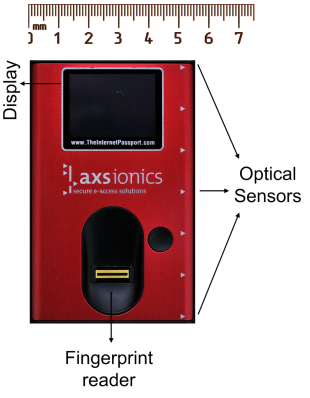
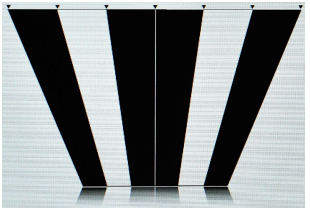


Fig. 1. Security token and its components

The token (see Fig. 1) features a $128 * 96$ pixel display, a fingerprint reader for local authentication of the user to the token through biometrics. The fingerprint reader also serves as a simple user input device for navigation and selection operations on the display. The token's size is that of a smart card, except that it is 5mm thick. It runs a custom firmware, which is verified during the boot process. All computations and storage operations are run within an EAL4+ certified ARM secure core CPU [ARM]. Finally, the token features optical sensors to read off data from a PC display.

Conceptually, the token server and the token constitute a secure remote procedure call (RPC) system. That is, the token server can call remote procedures (which are hardcoded in the firmware) on a token, the token then executes the designated procedure (which often involves user interaction with the token), and then finally returns the outcome of the computation to the server. The token server in turn makes this RPC functionality available as a secure service to third party applications.

Let us consider a standard usage example of the token system and assume that the user has established a session with an e-banking server using her PC. Then, in the course of the session the need for a secure computation arises, i.e., review and confirmation of e-banking transactions by the user. The request for performing the secure computation is sent from the e-banking server to the token server (e.g., using a secure Web Service). The token server then establishes a secure channel with the token specified in the request, sends the transaction details to the remote procedure on the token allowing the user to accept or reject the transaction, and finally returns the accept or reject choice to the initiating e-banking application.



(a) Freeze-image of animated flickering pattern.

(b) User holding token over flickering pattern (on PC display) to receive a message.

Fig. 2. Security token and optical flickering mechanism

The logical channel between the token server and a token features end to end security (i.e, confidentiality, authenticity, integrity, and freshness), through a proprietary protocol. The security properties of the channel essentially correspond to that of Internet security protocols, such as TLS and VPN. Physically, messages from the token server to the token are routed via the calling e-banking server to the user's PC to the token, which is "connected" (details see below) to the PC. Thereby the user PC plays the role of a network component only, which relays messages between the token and the server.

The physical transmission between the token and the user's PC can be established in two ways. One is using a USB cable. It features a high throughput in both directions, but requires the user to have a USB cable at hand and to connect it to the token and PC. The other way is unique to this token technology: it is based on an optical signal sent from the PC to the token. More precisely, an RPC request sent from the token server is encoded as a *flickering pattern* which is displayed in the PC's web browser (e.g., using GIF, Flash, or Java). The flickering (see 2(a)) is a rapidly alternating black and white pattern shown on a small area of the display. When the flickering appears on the PC's display, the user simply holds her card over the pattern (see Fig. 2(b)); the token then receives the request using its optical sensors. The bandwidth of the optical transmission channel is approximately 150bits/sec. Response messages from the token back to the server - if any - are shown on the token's display and then entered via the PC's keyboard by the user. The return messages are typically short cryptographic one-time transaction confirmation codes. The optical channel features an unparalleled connectivity (it works with any PC or Internet access device featuring a web browser and a sufficiently large display, without cabling) and ease of use (holding the token to the screen is an intuitive usage metaphor).

In our ad hoc attestation system described below, we don't use the token system to perform user and transaction authentication. We rather use it to securely send messages from the token server to a token (i.e., we don't send return messages to the server). In the following, we denote crypto processing (e.g., encrypting, MACing etc.) and encoding as flickering of a message m on the token server by $\text{fencode}(m, T_{ID})$, where T_{ID} denotes the identifier of the token to which the message is sent. Conversely, $m = \text{fdecode}(flickr)$ denotes the operation on the token that consists of decoding the flickering pattern and then crypto processing it (i.e, decrypting, MAC verification, etc.) to obtain the original message m.

4 Demonstrative Ad Hoc Attestation System

In the following we describe the implementation of a secure ad hoc attestation system, which realizes the usage paradigm of a *demonstrative* ad hoc attestation system, as discussed in §1. Our system makes use of the token technology described in the previous section.

In the following §4.1 we briefly review notions of trusted computing which we rely on in the description of our ad hoc attestation system in §4.2. Finally, in §4.3, we briefly discuss a variant of our ad hoc attestation protocol, which is secure under different assumptions from the ones given in §4.2.

4.1 Trusted Computing Basics and Notation

Let us briefly recall some of the Trusted Computing Group's (TCG) concepts used in our implementation. For a self-contained description, we refer to the TCG specifications and textbooks [Tru04, Tru05, Pea03, Mit05, Gra06, DC07]. At the heart of the TCG architecture is the Trusted Platform Module (TPM). This is a chip residing inside a platform, which performs trusted integrity measurement, storage, and integrity reporting operations. Each TPM has an *Endorsement Key*, which is a signing key whose public key is certified by a trusted third party (e.g., the TPM manufacturer). For privacy reasons, Endorsement Keys are used to obtain certificates on so called *Attestation Identity Keys (AIK)*, which are pseudonymous signing keys. To this end the TPM generates an AIK key-pair (AIK_{pub}, AIK_{priv}) and a certificate authority then issues a certificate on AIK_{pub}, vouching for the fact that the AIK key-pair was generated by a valid TPM. *Binding keys* are asymmetric encryption key-pairs. *Binding* is the operation of encrypting an object with the public key of a binding key. If the binding key is *non-migratable*, only the TPM that created the key can use its private key; hence, objects encrypted with a binding public key are effectively bound to a particular TPM. Finally, PCR registers are secure storage locations within a TPM, holding integrity measurements.

Next, we briefly describe the TPM functionality used by our protocol. The commands correspond to those available through the TCG Software Stack (TSS) [Trub]:

- createKey() generates an asymmetric binding key-pair (B_{pub}, B_{priv}), where the public key B_{pub} is returned to the caller and B_{priv} is a non-migratable private key stored inside the TPM.
- certifyKey(B_{pub}, AIK_{priv}, n) creates a certificate, which consists of a signature on the binding public key B_{pub} and a nonce n using the signing-key AIK_{priv}.
- quote$(AIK_{priv}, PCRdigest, n)$ signs a digest of selected PCR registers $PCRdigest$ and the nonce n using AIK_{priv}.
- bind(m, B_{pub}) encrypts a plaintext m under the binding public key B_{pub} and returns the resulting ciphertext. We note that bind() is executed within software (i.e., within TSS) and does not use TPM capabilities.
- unbind(E, B_{priv}) decrypts the cipher-text E, using the binding private key B_{priv}.

4.2 System Description

From a usage perspective our system works as follows: The user is equipped with a security token as described in §3. In a first step the user initiates the ad hoc attestation protocol by launching a corresponding program on the platform in question. Then, she holds her security token to the platform's display, to receive the outcome of the attestation using the flickering mechanism. Finally, the token displays whether the platform is trustworthy or not. We refer to this intuitive and easy usage metaphor as *demonstrative ad hoc attestation*.

The architecture of our demonstrative ad hoc attestation system consists of a token T, a target platform P (which is to be attested), and an attestation server S. On a high level the system works as follows: In a first step, S receives an attestation request (initiated by the user) from P, and then performs the actual attestation of P. The attestation of P by S is performed by running a variant of a remote attestation protocol, which we have tailored to fit our system. Then, in a second step, S securely reports the outcome of the attestation to the user's token T using the flickering mechanism of the token system. On a high level, "securely" means that our protocol assures that the flickering signal (and thus the outcome of the attestation) actually appears on the platform P on which the user has initiated the process.

Technically, we run on S an attestation engine and the token server component (described in §3). The advantage of this server based architecture is that the attestation engine can be managed centrally (see our discussion in §1 for details).

Here is the description of our demonstrative ad hoc attestation system:

Protocol 1 (Demonstrative ad hoc attestation system) . *Our demonstrative ad hoc attestation system consists of a user U, an attestation server S, a target platform P, and a security token T performing the following computational steps (see also Figure 3 for the protocol's message flow):*

1. *U initiates the ad hoc attestation protocol by launching the pre-installed ad hoc attestation component on P, and enters her token ID T_{ID}, as well as a nonce n_T generated by the token. This results in an attestation request* initiate(T_{ID}, n_T) *to S.*
2. *S randomly chooses a nonce n and sends n to P.*
3. *P computes:* $(B_{pub}, B_{priv}) =$ createKey(), $certB_{pub} =$ certifyKey(B_{pub}, AIK_{priv}, n), $quote =$ quote$(AIK_{priv}, PCRdigest, n)$, *and sends* $(certAIK_{pub}, B_{pub}, certB_{pub}, quote)$ *to S.*
4. *S verifies the validity of the digital signatures on $certAIK_{pub}$, $certB_{pub}$, $quote$; based on the integrity measurement contained in $quote$, S decides whether P is in a good state (*integrity$(P) \in \mathscr{G}$*) or bad state (*integrity$(P) \in \mathscr{B}$*). Now S performs the following steps: if* integrity$(P) \in \mathscr{G}$*), then let* $flickr =$ fencode$(\text{“}\texttt{good}''||n_T, T_{ID})$. *If* integrity$(P) \in \mathscr{B}$ *or if any of the initial signature verifications fails, let* $flickr =$ fencode$(\text{“}\texttt{bad}''||n_T, T_{ID})$*; finally* $E =$ bind$(flickr, B_{pub})$ *and sends E to P.*
5. *P computes* $flickr =$ unbind(E, B_{priv}) *and shows the flickering pattern $flickr$ on its display.*
6. *Once the flickering appears on P's display, U holds the token T to the flickering. Then, T runs* $m =$ fdecode$(flickr)$, *whereas either* $m = \text{“}\texttt{bad}''||n_T$ *or* $m = \text{“}\texttt{good}''||n_T$. *In the former case or when the nonce n_T does not match the nonce chosen in step* 1, *T outputs “*bad*” and “*good*” otherwise.*

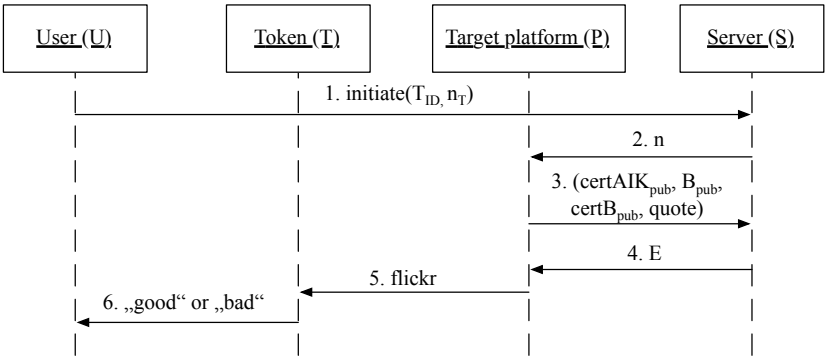


Fig. 3. Message flow of Protocol 1

We have implemented a working demonstrator of the above system in Java using the jTSS [JTS] library. Currently we assume, that the attestation component on the platform P is pre-installed. The attestation engine on the server S checks if the integrity measurements received from P match pre-defined good values. In a nutshell, the actual integrity reporting and evaluation done by our system is very basic.

Yet, one can easily enhance our system to use more sophisticated attestation techniques such as those in [RS04], property-based attestation [SS04], or quite likely also to future run-time attestation techniques [ESvD05]. Such enhancements will not change the above protocol structurally. In fact, the only changes that would result concern the measurements being sent in step 3 of the protocol, and how these measurements are evaluated by the attestation engine on the server in step 4.

Security analysis. In the following we discuss the security of our ad hoc attestation system with respect to the security goals set in §2. In security arguments of systems making use of Trusted Computing technologies, one typically needs to make some assumptions on the behavior of platforms P that are in a good state (i.e., integrity$(P) \in \mathscr{G}$), and so shall we. In fact, often the overly strong assumption is made that once integrity$(P) \in \mathscr{G}$ is established, P is deemed to be completely trustworthy. We refrain from this approach and prefer to more precisely describe the assumptions we make on P with integrity$(P) \in \mathscr{G}$.

First, we argue that if integrity$(P) \in \mathscr{B}$, then T does not output "`good`" (as we require in §2). Our assumption in the following is that a platform that is in good state can keep the flickering message $flickr$ it obtains in step 5 of the protocol private (at least for a period of time of a protocol execution). Now, we observe that, by construction of the protocol, a bad P won't get a valid $flickr$ message (protocol message 5) containing a "`good`" message in the first place. So P needs to steal a "`good`" $flickr$ message from a good platform P' (i.e., integrity$(P') \in \mathscr{G}$). Yet, P cannot replay such messages, because of the challenge n_T and because $flickr$ messages are integrity protected by the token system. So a bad P needs to get hold of a fresh "`good`" $flickr$ message that S sends to P'. Now, we observe that the usage of the binding key in the protocol asserts that only the good platform P' gets a fresh "`good`" $flickr$ message. The only possibility

for the attacker is thus to have a process running on P' that gets a fresh "good" $flickr$ message after it is decrypted on P' and then forwards it to P which then displays $flickr$. This is however impossible by our assumption that a good platform can keep flickering signals private.

Whether our assumption is realistic actually depends on the attestation mechanism being used and on properties of the platform P'. An attestation mechanism that can assert that all software running on P' is good, will be able to implement the assumption. On the other hand, more lightweight and thus more practical attestation techniques that only check a sub-set of platform measurements (e.g., the kernel and some key security sub-systems), might not be sufficient for main stream operating systems (e.g., Windows, OS X, Linux etc.). The reason is that these operating systems lack secure display functionalities (e.g., access control mechanisms to display contents). That is, any malicious process running on a main-stream OS can read the display contents of P', and thus obtain and forward $flickr$ to an impersonating platform. A possible remedy to this problem is to run an operating system on P', which provides secure display functionality [Ope, SS05, SVNC04].

It remains to argue that if integrity$(P) \in \mathscr{G}$, then T does not output "bad". In this case an attacking bad platform P' (i.e., integrity$(P') \in \mathscr{B}$) can easily get hold of a fresh "bad" $flickr$ message. Now, all the attacker needs to do is to have a process running on P that will display the "bad" $flickr$ message. One could argue that, since integrity$(P) \in \mathscr{G}$, it is impossible for the attacker to mount this attack on P. On the other hand, only a very thorough attestation mechanism can assure the absence of such processes. Anyway, we consider this security property less important than the one discussed before, since it "only" concerns a denial of service condition, where a user refrains from using a good platform because the attacker tricks him to belief that the platform is bad.

A potential weakness of our system results from non-software attacks, where an attacker films the flickering pattern shown on P's display, and relays it to an impersonating platform. This is clearly a time-consuming and thus rather expensive attack, which probably only becomes relevant in "high security" scenarios.

4.3 Sketch of an Alternative Protocol

In this section we sketch a variant of our ad hoc attestation protocol, whose security relies on different assumptions than those in the previous section. The architecture underlying the protocol remains the same, while we additionally require that the target platform is equipped with a fingerprint reader. From the usage point of view the protocol is very similar to the one above, expect that at a certain point during the protocol, the user additionally has to scan her fingerprint on the target platform. For space reasons we only give a sketch of the protocol:

- Steps 1 - 3 are the same as in Protocol 1, except that P in step 3, instead of the binding keys, generates a pair of signing keys (U_{pub}, U_{priv}) and sends U_{pub} along with a certificate $certU_{pub}$ on U_{pub} to S.
- In step 4, in a analogy to Protocol 1, S first checks the various signatures. Then, if integrity$(P) \in \mathscr{B}$, set $flickr =$ fencode(''bad'', T_{ID}). On the other hand, if integrity$(P) \in \mathscr{G}$, then S chooses a challenge n_F and sends it to P.

- In step 5, P asks the user to scan her fingerprint r using the fingerprint reader built into P. Then P sends r and $\sigma_r = \text{sign}(r||n_R, U_{priv})$ to S.
- In step 6, S checks if the signature σ_r on $r||n_R$ is valid and the freshness of n_R. If so it sets $flickr = \text{fencode}(\text{``conditional good''}||r, T_{ID})$; otherwise, it sets $flickr = \text{fencode}(\text{``bad''}, T_{ID})$ and sends $flickr$ to P.
- In step 7, P shows the flickering $flickr$ on its display.
- In step 8, when the flickering appears on P's display, the user holds her token T to the flickering. Then, T runs $m = \text{fdecode}(flickr)$. If $m = \text{``bad''}$, $T$ outputs "bad". Otherwise, if $m = \text{``conditional good''}||r$, T checks if r is the fingerprint of the token owner U. To this end, it either retrieves a pre-stored fingerprint r_U or using the token's fingerprint reader acquires a fingerprint r_U, and then matches r_U against r. If the fingerprints match, T outputs "good", and "bad" otherwise.

Let us briefly discuss the difference between Protocol 1 and the one here with respect to their security properties. The difference is in the assumptions made on a good platform P (i.e., integrity$(P) \in \mathscr{G}$). In the previous protocol we have assumed that such P can keep $flickr$ confidential, to show that if integrity$(P) \in \mathscr{B}$, then T does not output "good". The assumption underlying this property is different for the protocol in this section. On a high-level, the assumption we need is that P can preserve the integrity of a fingerprint measurement it carries out. In fact, the signature in step 5 asserts that the fingerprint measurement r actually originates from the same platform P that has been attested. Now assume that a P with integrity$(P) \in \mathscr{G}$ in step 3 always reports a fresh and correct fingerprint measurement. By the verifications in step 6, it thus follows that the platform that has scanned U's fingerprint is the one that has been attested by S. This implies that the protocol is secure against platform in the middle attacks.

Moreover, the protocol here is not susceptible against the "flickering filming" attack described above.

5 Related Work

There is a body of work on remote attestation [RS04, SS04, GPS06, FSE06, GSS+07]. Yet, as discussed in §1, remote attestation protocols do not solve the ad hoc attestation problem. Several research works have considered the authentication of platforms in a setting where the user is equipped with a trustworthy device [ABKL93, SA99, ADSW99], [CYCY00, DBW02, MPR05]. On a high level, the goal of these works is the same as that of ad hoc attestation: the user wants to assure that the terminal she is going to use is trustworthy. However, these works assume that the legitimate terminals being authenticated are tamper-resistant and thus trustworthy. That is, they do not address ad hoc attestation problem.

Yet, there are several ideas in previous works which appear in ours. Our usage of an attestation server is inspired by the usage of an authentication servers common in the works of [ABKL93, ADSW99, CYCY00]. Another idea we have adapted is that of *demonstrative identification* [SA99, DBW02, MPR05] paradigm, where a user identifies a platform by establishing physical contact or proximity with the platform in question. This paradigm is highly intuitive, and underlies our demonstrative ad hoc attestation paradigm.

Very recent papers [MPSvD07, SG07] outline the open challenges of (what we call) ad hoc attestation, and thus have motivated our work. As discussed in detail in §1, we solve most of the outlined challenges. While [MPSvD07] only outlines open challenges, Garriss et al [SG07] actually propose and implement a concrete ad hoc attestation protocol with the user device being a Bluetooth enabled smart phone. This is the only ad hoc attestation protocol in the literature we are aware of. Like us, they also consider using a central attestation server. Yet, they do not solve the problem of platform in the middle attacks. Besides that, our demonstrative ad hoc attestation paradigm is new, and due to its closed execution architecture of our user device, our solution can offer higher security guarantees than those using a smart phone.

Some of the works mentioned, e.g., [ADSW99, CYCY00], use techniques to assert that the platform being authenticated is the one located in front of the user. These techniques are similar to our approach of sending a flickering signal to the attested platform (see §4.2). On the other hand, our approach of reading a user's fingerprint (see §4.3) to locate a platform is, to the best of our knowledge, new.

Distance bounding techniques [DM07] could be used to assert that the machine being ad hoc attested is within a certain physical perimeter. However, we currently believe this might only work when the attestation and distance bounding is performed by the portable user device and will not work with our server based scenario, where the server is located remotely.

6 Conclusion and Future Work

We have described the implementation of an intuitive and easy to use ad hoc attestation system. The current state of our implementation is that of working research demonstrator. In future work, we plan to evolve this prototype into a practically usable product demonstrator. To this end we plan to replace the current Java component which needs to be installed on the target platform. The envisioned approach is to use a web browser, where the user enters the URL of the attestation server, which then sends the signed Java applet to the target-platform. This minimizes the code to be pre-installed on the target platform and improves ease of use by employing a browser. Second, we plan to integrate our ad hoc attestation system with the Turaya secure OS [TUR], which features suitable attestation functionality. Combining Turaya with our ad hoc attestation system results in a system, which is interesting for environments and organizations with high security needs.

References

[ABKL93] Abadi, M., Burrows, M., Kaufman, C., Lampson, B.: Authentication and delegation with smart-cards. In: TACS 1991: Selected papers of the conference on Theoretical aspects of computer software, Netherlands, pp. 93–113. Elsevier Science Publishers, Amsterdam (1993)

[ADSW99] Asokan, N., Debar, H., Steiner, M., Waidner, M.: Authenticating public terminals. Comput. Networks 31(9), 861–870 (1999)

[ARM] Arm secure core processor family, http://www.arm.com/products/cpus/families/securcorefamily.html

[AXS] Axsionics homepage, http://www.axsionics.com/

[CJM07] Boneh, D., Jackson, C., Mitchell, J.C.: Transaction generators: Rootkits for the web. In: Proceedings of the Workshop on Hot Topics in Security (HotSec) (2007)

[CYCY00] Cheng, K.S.C.Y., Yunus, J.: Authentication public terminals with smart cards. In: TENCON 2000, 24-27 September 2000, vol. 1, pp. 527–530 (2000)

[DBW02] Stewart, P., Balfanz, D., Smetters, D.K., Chi, H.: Talking to strangers: Authentication in ad-hoc wireless networks. In: Symposium on Network and Distributed Systems Security (NDSS 2002) (2002)

[DC07] Catherman, R., Safford, D., van Doorn, L., Challener, D., Yoder, K.: A Practical Guide to Trusted Computing. IBM Press (2007)

[DM07] Drimer, S., Murdoch, S.J.: Keep your enemies close: Distance bounding against smartcard relay attacks. In: USENIX Security Symposium (August 2007)

[ESvD05] Perrig, A., Shi, E., van Doorn, L.: Bind: A time-of-use attestation service for secure distributed systems. In: Proceedings of IEEE Symposium on Security and Privacy (May 2005)

[FSE06] Röder, P., Stumpf, F., Tafreschi, O., Eckert, C.: A robust integrity reporting protocol for remote attestation. In: Proceedings of the Second Workshop on Advances in Trusted Computing (WATC 2006 Fall) (December 2006)

[GPS06] Goldman, K., Perez, R., Sailer, R.: Linking remote attestation to secure tunnel endpoints. In: STC 2006: Proceedings of the first ACM workshop on Scalable trusted computing, pp. 21–24. ACM, New York (2006)

[Gra06] Grawrock, D.: The Intel Safer Computing Initiative. Intel Press (2006)

[GSS+07] Gasmi, Y., Sadeghi, A.-R., Stewin, P., Unger, M., Asokan, N.: Beyond secure channels. In: STC 2007: Proceedings of the 2007 ACM workshop on Scalable trusted computing, pp. 30–40. ACM, New York (2007)

[JTS] http://trustedjava.sourceforge.net/

[Kre] Krebs,B.: Banks: Losses from computer intrusions up in (2007)

[Mit05] Mitchell, C. (ed.): Trusted Computing. The Institution of Electrical Engineers (2005)

[MPR05] McCune, J.M., Perrig, A., Reiter, M.K.: Seeing-is-believing: Using camera phones for human-verifiable authentication. In: SP 2005: Proceedings of the 2005 IEEE Symposium on Security and Privacy, pp. 110–124. IEEE Computer Society, Washington (2005)

[MPSvD07] McCune, J.M., Perrig, A., Seshadri, A., van Doorn, L.: Turtles all the way down: Research challenges in user-based attestation. In: Proceedings of the Workshop on Hot Topics in Security (HotSec) (2007)

[Ope] Open Trusted Computing, http://www.opentc.net

[Pea03] Pearson, S. (ed.): Trusted Computing Platforms: TCPA Technology in Context. Prentice Hall, Englewood Cliffs (2003)

[Ran04] Ranganathan, K.: Trustworthy pervasive computing: The hard security problems. In: PERCOMW 2004: Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops, p. 117. IEEE Computer Society, Washington (2004)

[RS04] Jaeger, T., van Doorn, L., Sailer, R., Zhang, X.: Design and implementation of a tcg-based integrity measurement architecture. In: SSYM 2004: Proceedings of the 13th conference on USENIX Security Symposium. USENIX Association, Berkeley (2004)

[SA99] Stajano, F., Anderson, R.J.: The resurrecting duckling: Security issues for ad-hoc wireless networks. In: Proceedings of the 7th International Workshop on Security Protocols, London, UK, pp. 172–194. Springer, Heidelberg (1999)

[SG07] Berger, S., Sailer, R., van Doorn, L., Zhang, X., Garriss, S., Caceres, R.: Towards trustworthy kiosk computing. In: Proc. of 8th IEEE Workshop on Mobile Computing Systems and Applications (HotMobile) (February 2007)

[SS04] Sadeghi, A.-R., Stüble, C.: Property-based attestation for computing platforms: caring about properties, not mechanisms. In: NSPW 2004: Proceedings of the 2004 workshop on New security paradigms, pp. 67–77. ACM, New York (2004)

[SS05] Sadeghi, A.R., Stüble, C.: Towards Multilaterally Secure Computing Platforms - With Open Source and Trusted Computing. Elesevier 10, 83–95 (2005)

[SVNC04] Shapiro, J.S., Vanderburgh, J., Northup, E., Chizmadia, D.: Design of the eros trusted window system. In: SSYM 2004: Proceedings of the 13th conference on USENIX Security Symposium, p. 12. USENIX Association, Berkeley (2004)

[Trua] Trusted Computing Group (TCG). About the TCG, http://www.trustedcomputinggroup.org/about/

[Trub] Trusted Computing Group (TCG). TSS specifications, https://www.trustedcomputinggroup.org/groups/software/

[Tru04] Trusted Computing Group. TCG Architecture Overview (April 2004)

[Tru05] Trusted Computing Group (TCG). TPM Main Specification 1.2, Rev. 85 (February 2005), https://www.trustedcomputinggroup.org/groups/tpm/

[TUR] Turaya OS homepage, http://www.emscb.com/content/pages/turaya.htm

Property-Based Attestation without a Trusted Third Party

Liqun Chen[1], Hans Löhr[2], Mark Manulis[3], and Ahmad-Reza Sadeghi[2]

[1] HP Laboratories, Bristol, UK
liqun.chen@hp.com
[2] Horst Görtz Institute for IT Security, Ruhr-University of Bochum, Germany
{hans.loehr,ahmad.sadeghi}@trust.rub.de
[3] UCL Crypto Group, Université Catholique de Louvain, Belgium
mark.manulis@uclouvain.be

Abstract. The Trusted Computing Group (TCG) has proposed the binary attestation mechanism that enables a computing platform with a dedicated security chip, the Trusted Platform Module (TPM), to report its state to remote parties. The concept of property-based attestation (PBA) improves the binary attestation and compensates for some of its main deficiencies. In particular, PBA enhances user privacy by allowing the trusted platform to prove to a remote entity that it has certain properties without revealing its own configuration.

The existing PBA solutions, however, require a Trusted Third Party (TTP) to provide a reliable link of configurations to properties, e.g., by means of certificates. We present a new privacy-preserving PBA approach that avoids such a TTP. We define a formal model, propose an efficient protocol based on the ideas of ring signatures, and prove its security. The cryptographic technique deployed in our protocol is of independent interest, as it shows how ring signatures can be used to efficiently prove the knowledge of an element in a list without disclosing it.

Keywords: Property-based attestation, user privacy, ring signatures, proof of membership, configuration anonymity.

1 Introduction and Background

A fundamental issue in interaction between computing platforms is "trust" or "trustworthiness" — whether a remote platform behaves in a reliable and predictable manner, or will be (or already has been) subject to subversion. Cryptographic mechanisms support the establishment of secure channels and authorized access, but without assurance about the integrity of the communication endpoints. Commodity computing platforms suffer from inherent vulnerabilities due to high complexity, and lack of efficient protection against tampering or malware. Hence, an important subject of current research is to develop mechanisms for gaining assurance about the trustworthiness of remote peers regarding their

T.-C. Wu et al. (Eds.): ISC 2008, LNCS 5222, pp. 31–46, 2008.

integrity, platform configuration, and security policies. The concept of Trusted Computing aims at resolving such issues.

The TCG approach and binary attestation. An industrial approach towards the realization of the Trusted Computing functionality within the computing platforms is the initiative of the Trusted Computing Group (TCG). The TCG has published many specifications amongst which the most important one is that of the Trusted Platform Module (TPM) [25]. Currently, TPMs are implemented as small, tamper-evident hardware modules embedded in commodity platforms, providing (i) a set of cryptographic functionalities, (ii) the protection of cryptographic keys, (iii) the authentication of platform configuration (attestation), and (iv) cryptographic sealing of sensitive data to particular system configurations. However, the TCG defines only a limited set of commands, and the firmware cannot be programmed by end-users to execute arbitrary functions. Millions of platforms (PCs, notebooks, and servers) being sold today are equipped with TPMs.

One of the main features supported by the TPM is the so-called trusted integrity measurement: a hash value of the platform state is computed during the boot process and stored in specific registers of the TPM, the *Platform Configuration Registers* (PCRs),those state is also called the platform's *configuration*. Of potential interest is the offered functionality called *binary attestation*, which allows a remote party (verifier) to get an authentic report about the binary configuration of another platform (prover), given by the prover's TPM signature on the configuration.

Deficiencies of TCG binary attestation. TCG binary attestation suffers from several shortcomings: The slightest change in the measured software or configuration files — whether security-relevant or not — will lead to a changed binary configuration. In general, it is not clear, how a verifier should derive the trustworthiness of a platform from such a binary value. System updates and backups are highly non-trivial; the multitude of different versions of many pieces of software cause serious manageability problems.

From the privacy point of view, binary attestation bears several risks: (1) The TPM's public key needed to verify an attestation could be used to identify a TPM and trace a platform. To solve this problem, Brickell et al. [3] introduced the *Direct Anonymous Attestation* (DAA) protocol. Improvements of DAA and alternative DAA schemes (e.g., [5,4,6]) are orthogonal to our work and could be used as a building block for our protocol. (2) Typically the information about the configuration of a computing platform or application is revealed to a remote party requesting the state of a platform. This information can be misused to discriminate against certain configurations (for example, operating systems) and even vendors, or may be exploited to mount attacks.

Property-based attestation. One general concept to overcome shortcomings of the TCG's binary attestation is to transform the binary attestation into the *property-based attestation* (PBA), as described by Sadeghi and Stüble [21], and by Poritz et al. [19]. The basic idea of PBA requires a computing platform to

attest that it fulfills the desired (security) requirements, so-called 'properties', without revealing a respective software or/and hardware configuration. The formal definition of properties as well as the development of various practical solutions for PBA are still active areas of ongoing research.

One concrete solution for PBA was proposed by Chen et al. in 2006 [11]. Their protocol requires an off-line Trusted Third Party (TTP) to publish a list of trusted configurations and respective certificates which attest that the configurations provide specific properties. A prover can use the signed configurations and certificates to prove to a verifier that it has appropriate configurations associated with the certified properties, without disclosing the specific configurations, which the platform holds.

Another solution for PBA is proposed by Kühn et al. [14]. In their work, the authors suggest a modified system boot architecture, such that not binary hash values of files are stored by the TPM, but instead abstract values representing properties, e.g., a public key associated with a property certificate. However, this approach also requires a TTP to issue certificates for properties and the bootloader must be binary-attested.

The drawback of these solutions is that such a TTP might not be available or/and desirable in many real applications, for example if two entities/users want to have a private communication with each other. They have their own understanding of the relation between various configurations and security properties. They do not need (and do not want) to ask any kind of TTPs to certify a correlation between the configurations and properties. However, they still want to keep their platform configuration information secret from each other.

Our contribution. In this paper, we propose a protocol for PBA that does not require the involvement of a TTP to certify properties, where a platform (equipped with a TPM) convinces a remote party that its configuration satisfies a given property. For this, the two parties first agree on a set of trusted configuration specifications, which they both consider to be trustworthy, i.e., associated with a well-defined security property or properties. The platform then proves that its configuration specification is in this set. In our protocol, TPM and the host software compute the proof jointly.

For some applications, it might be unrealistic to assume that the parties in the attestation protocol can decide themselves which configurations are trustworthy and which are not, and thus they still have to rely on third parties in practice. Our protocol has the advantage that even in this case no global trusted party is necessary: both participants can choose independently how to agree on trustworthy configurations or they can delegate this decision to other parties.

Further, we define a formal security model for PBA, which we also use in our proofs, and where the main security requirements are evidence authentication and configuration privacy. While the former guarantees an unforgeable binding between the platform and its configuration specification, the latter provides the non-disclosure of the configuration specification. In our PBA protocol, these requirements are achieved through the use of a ring signature (cf. Section 4.3),

i.e. configuration privacy results from the anonymity of the signer whereas evidence authentication is based on the unforgeability of the signature.

Moreover, the cryptographic technique employed in our protocol may be of independent interest: We show how ring signatures can be used for efficiently proving the knowledge of an element in a list without disclosing it.

Outline. In Section 2, we introduce the system model of property-based attestation. In Section 3, we sketch different solutions on a high level. In Section 4, we set up notation and explain some building blocks which will be used in our concrete PBA scheme. In Section 5, we present and discuss a new PBA scheme, and in Section 6 we define a formal security model and state theorems about the security of the scheme. In Section 7, we conclude the paper by mentioning some unsolved problems and future work.

2 System Model for PBA

The following system model for PBA will serve as the basis for the security model in Section 6.

Involved parties. A PBA protocol involves two participants: a *prover* $\mathcal{P}$ and a *verifier* $\mathcal{V}$. The prover is a platform consisting of a host $\mathcal{H}$ and a trusted module TPM $\mathcal{M}$ (see Figure 1). To cover multiple executions of the protocol we consider multiple instances and use indices to distinguish among their participants, i.e., $\mathcal{P}_i$, $\mathcal{V}_i$. Each instance includes a single protocol execution with some *unique session identity* (SID) and two participants $\mathcal{P}_i$ and $\mathcal{V}_j$ are treated as communication partners (in the same instance) if they share the same SID.

Assumptions. It is assumed that the communication between a host $\mathcal{H}_i$ and its TPM $\mathcal{M}_i$ is through a secure channel (private and authentic), and that $\mathcal{M}_i$ and $\mathcal{V}_i$ communicate via $\mathcal{H}_i$. We omit the indices i and j of the participants in an instance when no risk of confusion exists. Moreover, the TPM is trusted by all parties and possesses a secret (signing) key $sk_{\mathcal{M}}$ which is unknown to the host. The corresponding public (verification) key is available to both $\mathcal{P}$ and $\mathcal{V}$; see also "trust relations" in Section 6.1.

Properties and configurations. Each prover $\mathcal{P}$ has a configuration value denoted $cs_{\mathcal{P}}$, which is an authenticated record about its platform's configuration. The value $cs_{\mathcal{P}}$ is known to both the host $\mathcal{H}$ and TPM $\mathcal{M}$, and it is computed by $\mathcal{M}$ from correctly measured configuration information, stored securely in special-purpose registers — the platform configuration registers (PCRs). As a result, $\mathcal{H}$ cannot modify this value without being detected. This is guaranteed by the properties of secure measurement and reporting based on the trusted computing technology [25]. It is assumed that before running the PBA protocol, $\mathcal{P}$ and $\mathcal{V}$ have already agreed on a set of configuration values denoted $CS = \{cs_1, ..., cs_n\}$ that satisfy the same property. So, we say that a configuration value cs satisfies a given property associated with CS, if and only if $cs \in CS$.

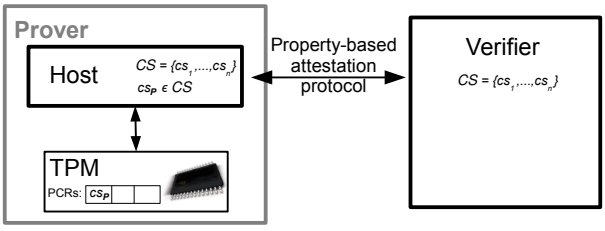


Fig. 1. PBA system model

Definition of PBA. A property-based attestation (PBA) scheme consists of the following three polynomial-time algorithms:

- Setup: Given the security parameter 1^κ, this probabilistic algorithm selects a set of public parameters that are necessary to run the PBA protocol, and produces a private/public key pair for each TPM.
- PBA-Sign: On input a configuration value $cs_\mathcal{P}$, a list of admissible configurations CS, and a nonce N_v, this (distributed) randomized algorithm outputs a signature σ on $cs_\mathcal{P}$.
- PBA-Verify: On input a candidate signature σ and CS, this deterministic algorithm outputs 1 (accept) if σ is a valid signature on a value from CS, or 0 (reject) otherwise.

3 Solutions

In this section, we sketch two high-level solutions for PBA without relying on trusted parties to certify the link between configurations and properties.

Basically, $\mathcal{P}$ has to prove that its configuration value $cs_\mathcal{P}$ belongs to the agreed set $CS = \{cs_1, \ldots, cs_n\}$. More precisely, $\mathcal{V}$ would accept a proof if and only if: (i) The proof is created by a *valid* TPM. If TPM anonymity is required, the DAA scheme [3] can be used to provide this feature. (ii) The proof is a fresh response to a specific challenge from $\mathcal{V}$. (iii) The proof ensures that $cs_\mathcal{P} = cs_j$ for an index $j \in \{1, 2, \ldots, n\}$, but does not reveal the value of j.

Such a proof implements PBA-Sign, whereas PBA-Verify is the verification of the proof. In Setup, the keys for the TPM and system parameters are generated.

Solution 1: *TPM as single signer.* The proof can be achieved by a new TPM command defined as follows:

1. TPM takes as input a list of configurations CS and a nonce N. The nonce is assumed to be chosen by the verifier $\mathcal{V}$.
2. TPM checks for each $cs_j \in CS$ if $cs_\mathcal{P} = cs_j$, until either a match is found, or the entire list has been checked.
3. If $cs_\mathcal{P}$ is in the list, the TPM generates a signature on $(1, N, CS)$; otherwise, the TPM generates a signature on $(0, N, CS)$, which is then forwarded to $\mathcal{V}$.

The obvious drawbacks of this approach are: TPM operations depend on the size n of CS ($\mathcal{O}(n)$ in a straightforward implementation, and $\mathcal{O}(\log n)$ if CS is a sorted list). As the TPM's memory is very limited, this would either impose a severe restriction on the size of CS, or the transfer of the list would have to be split up, causing further complexity of the TPM-command and slowing down the communication between host and TPM, due to the overhead.

Solution 2: *TPM shares signer role with host.* In this solution, the TPM signs a hidden version — a commitment — of the configuration $cs_{\mathcal{P}}$, and the host completes the proof that the hidden configuration is in the set CS. A similar approach is used in the DAA protocol [3].

Our PBA protocol proposed in Section 5 is an elegant and efficient example of this solution. It makes use of ring signatures in that the host computes n public keys for a ring signature scheme from the configurations in CS and the commitment to $cs_{\mathcal{P}}$ (which was signed by the TPM), and determines the secret key that corresponds to $cs_{\mathcal{P}}$. The signer anonymity of the ring signature scheme ensures that the verifier does not learn which key has been used for signing, thus $cs_{\mathcal{P}}$ is not disclosed. Our construction guarantees that the prover succeeds only if the hidden configuration $cs_{\mathcal{P}}$ is indeed in CS.

Current TPMs support all operations (random number generation, modular exponentiation, and signature generation) needed by our protocol. However, the TCG currently does not specify a command to create and sign a commitment to a configuration which is stored inside the TPM. To implement such a command, only firmware changes would be required.

Other protocols for similar solutions could be developed, for instance based on existing zero-knowledge proofs (e.g., [8,13,7]) or zero-knowledge sets [15].

4 Preliminaries

4.1 TPM Signatures

The existing TCG technology defines two ways for a TPM to create own digital signature $\sigma_{\mathcal{M}}$. The first way is to use DAA [3]. With a DAA signature, a verifier is convinced that a TPM has signed a given message, but the verifier cannot learn the identity of the TPM. The message to be signed can be either an Attestation Identity Key (AIK), or an arbitrary data string, The second way is to use an ordinary signature scheme. A TPM generates a signature using an AIK as signing key, which could either be certified by a Privacy-CA, or it could be introduced by the TPM itself using a DAA signature. For simplicity, we do not distinguish these two cases, and denote by $\sigma_{\mathcal{M}} := \mathtt{SignM}(sk_{\mathcal{M}}; m)$ the output of TPM's signing algorithm on input the TPM's signing key $sk_{\mathcal{M}}$ and a message m, and by $\mathtt{VerM}(vk_{\mathcal{M}}; \sigma_{\mathcal{M}}, m)$ the corresponding verification algorithm, which on input the TPM's verification key $vk_{\mathcal{M}}$ outputs 1 if $\sigma_{\mathcal{M}}$ is valid and 0 otherwise.

4.2 Commitment Scheme

We apply the commitment scheme by Pedersen [18]: Let $sk^m_{\mathtt{com}}$ be the secret commitment key. A commitment on a message m is computed as $C_m := g^m h^{sk^m_{\mathtt{com}}}$

mod P. P is a large prime, h is a generator of a cyclic subgroup $G_Q \subseteq \mathbb{Z}_P^*$ of prime order Q and $Q|P-1$. g is chosen randomly from $\langle h \rangle$; furthermore, $\log_h(g)$ is unknown to the committing party. Both the message m and $sk_{\mathtt{com}}^m$ are taken from $\mathbb{Z}_Q$. The Pedersen commitment scheme as described above is perfectly hiding and computationally binding, assuming the hardness of the discrete logarithm problem in a subgroup of $\mathbb{Z}_P^*$ of prime order (for P prime).

4.3 Ring Signatures

The notion of a ring signature was first introduced by Rivest et al. [20]. It allows a signer to create a signature with respect to a set of public keys. Successful verification convinces a verifier that a private key corresponding to one of the public keys was used, without disclosing which one. In contrast to group signatures, no group manager is needed.

For various security definitions for ring signatures see [2]. Recent efficient ring signature schemes which are provably secure in the standard model (i.e., without using random oracles) are proposed in [23,9], where in [9] a signature with size only $\mathcal{O}(\sqrt{n})$ is proposed. Dodis et al. [12] showed that ring signatures with constant size in the number of public keys can be achieved in the random oracle model.

Unfortunately, none of these schemes can be used easily for our purposes: In our protocol, we employ a construction, where the public keys for the ring signature are computed from commitments formed by the TPM. We show how this can be done efficiently for Pedersen commitments (cf. Section 4.2) and public keys of the form $y = g^x \bmod P$, where x is the corresponding secret key. However, the schemes above use keys of different types.

In Figure 2, we recall an efficient ring signature scheme from [1], which we propose to use for our PBA solution. The scheme is a generalization of the Schnorr signature scheme [22]: Intuitively, the product in step 2(b) corresponds to combined commitments for individual Schnorr signatures, in step 2(c) and 2(d), the challenges for the individual Schnorr signatures are derived from a single challenge, and in step 2(e), the secret key is used to compute s. The verification equation, where the sum of the challenges is compared to a hash value, ensures that a valid signature cannot be created without a secret key x_j. The scheme is provably secure in the random oracle model, under the discrete logarithm assumption.

We denote the generation of a ring signature σ_r on message m with respect to the public key ring $\{y_i\}_{1 \le i \le n}$ and with private signing key x by $\sigma_r := \mathtt{SigRing}(x; \{y_i\}; m)$. Signature verification is denoted by $\mathtt{VerRing}(\{y_i\}; \sigma_r, m)$. For simplicity, we omit the public parameters g, P, Q and the range of the index i in our notation.

5 Ring Signature-Based PBA without TTP

In this section, we propose a protocol for PBA, which is based on ring signatures. The TPM generates a signature on a commitment to the configuration $cs_{\mathcal{P}}$. Then

1. Key generation. Let κ be a security parameter. On input 1^κ, create g, P and Q. A signer S_i $(i = 1, ..., n)$ chooses $x_i \in_R \{0,1\}^{\ell_Q}$ and compute $y_i = g^{x_i} \bmod P$. Output its public key (g, P, Q, y_i) and the corresponding secret key x_i.
2. Signing algorithm $\texttt{SigRing}(x_j; \{y_i\}; m)$.
   A signer who owns secret key x_j generates a ring signature on a message m with public key list (g, P, Q, y_i) $(i = 1, ..., n)$, where $j \in \{1, ..., n\}$ as follows:
   (a) Choose $\alpha, c_i \in_R \{0,1\}^{\ell_Q}$ for $i = 1, ..., n$, $i \neq j$.
   (b) Compute $z = g^\alpha \prod_{i=1, i\neq j}^{n} y_i^{c_i} \bmod P$.
   (c) Compute $c = \texttt{Hash}(g \| P \| Q \| y_1 \| ... \| y_n \| m \| z)$.
   (d) Compute $c_j = c - (c_1 + ... + c_{j-1} + c_{j+1} + ... + c_n) \bmod Q$.
   (e) Compute $s = \alpha - c_j \cdot x_j \bmod Q$.
   (f) Output the signature $\sigma_r = (s, c_1, ..., c_n)$.
3. Verification algorithm $\texttt{VerRing}(\{y_i\}; \sigma_r, m)$.
   To verify that the tuple $\sigma_r = (s, c_1, ..., c_n)$ is a ring signature on message m, check that $\sum_{i=1}^{n} c_i \equiv \texttt{Hash}(g \| P \| Q \| y_1 \| ... \| y_n \| m \| g^s y_1^{c_1} ... y_n^{c_n} \bmod P)$.

Fig. 2. A Ring Signature Scheme [1]

the host $\mathcal{H}$ creates a proof, using a ring signature, that $cs_\mathcal{P}$ is in the agreed set CS of configurations with the given property. The verifier $\mathcal{V}$ verifies the TPM signature and the ring signature.

Note that in our protocol, the TPM is trusted by all parties, but its resources are restricted, and it can execute only a very limited set of instructions. The host $\mathcal{H}$ is not trusted by the verifier $\mathcal{V}$, hence the protocol has to protect evidence authentication against a malicious host. $\mathcal{H}$ cannot be prevented from disclosing its own configuration $cs_\mathcal{P}$, thus for configuration privacy, we have to assume that $\mathcal{H}$ is honest.

5.1 Security Parameters

We suggest the following security parameters (values in parentheses indicate realistic values[1] for current TPMs):

- ℓ_{cs} (160): the size of the value of $cs_\mathcal{P}$.
- $\ell_\varnothing$ (160): the security parameter for the anti-replay value (nonce).
- ℓ_P (1024): the size of the modulus P.
- ℓ_Q (160): the size of the order Q of the subgroup of $\mathbb{Z}_P^*$.

The parameters ℓ_P and ℓ_Q should be chosen such that the discrete logarithm problem in the subgroup of $\mathbb{Z}_P^*$ of order Q with P and Q being primes such that $2^{\ell_Q} > Q > 2^{\ell_Q - 1}$ and $2^{\ell_P} > P > 2^{\ell_P - 1}$, is computationally hard.

[1] Examples based on the use of SHA-1 [16] as a hash function (like in current TPMs), and recommendations of the US National Institute of Standards and Technology (NIST) for similar applications (see, for instance, [17]); changes corresponding to stronger hash-functions, such as SHA-256, can be made straightforwardly.

5.2 Setup

We assume that $\mathcal{V}$ can verify TPM signatures (including revocation verification) and that $\mathcal{H}$ and $\mathcal{V}$ have agreed on a set of configurations CS.

Prior to the execution of the PBA protocol, the parties have to agree on the following parameters, which can be used for several protocol runs (potentially with different sets CS): primes P and Q, generators g and h of a subgroup of $\mathbb{Z}_P^*$ of order Q (i.e., the discrete logarithm problem is hard in $\langle g \rangle = \langle h \rangle$). The discrete logarithm $\log_g(h) \bmod P$ must be unknown to $\mathcal{H}$.

5.3 Signing and Verifying Protocol

The attestation procedure executed between a TPM ($\mathcal{M}$), its host ($\mathcal{H}$), and a verifier ($\mathcal{V}$) is described in Figure 3. As a result of the protocol, the host creates a ring signature σ_r, which is based on a TPM signature σ_M on the message C, which is a commitment to $cs_\mathcal{P}$. The TPM has to create and sign C, which it then opens towards $\mathcal{H}$. To create the ring signature, the host uses the value r as the secret key (if $cs_\mathcal{P} \in CS$, this works, because $y_j = h^r \bmod P$ for some j). From the ring signature, the verifier is convinced that the platform has been configured with one of the set of acceptable configuration specifications, $CS = \{cs_1, \cdots, cs_n\}$, without knowing which one.

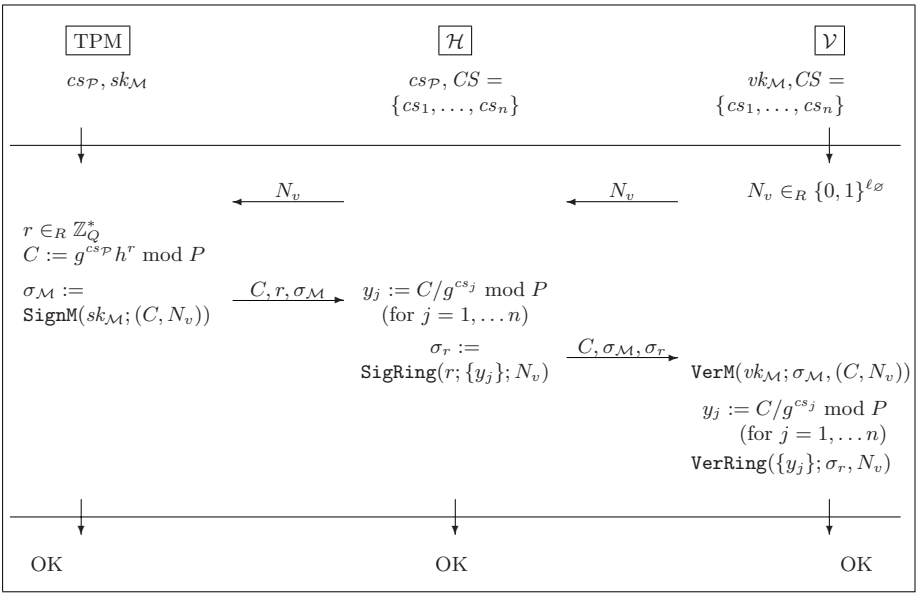


Fig. 3. The protocol of the PBA scheme. Common input: g, h, P, Q.

5.4 Protocol Properties

Our protocol has some interesting properties:

First, no trusted third party is needed for this protocol. The only exception is the certification of TPM keys: The verifier may rely on a DAA issuer or a Privacy-CA to ensure that the TPM key belongs to a valid TPM, depending on the TPM signature scheme (see Section 4.1). However, this is completely independent from the PBA protocol, and neither a DAA issuer nor a Privacy-CA could breach the configuration privacy of our protocol.

Second, the configuration set CS is created flexibly, dependent on the agreement between prover $\mathcal{P}$ and verifier $\mathcal{V}$. One approach to negotiate the set of acceptable configurations could be analogous to the SSL/TLS handshake: The prover sends a proposal for CS to $\mathcal{V}$, who can then select an appropriate subset. However, our protocol allows for different ways to agree on CS; the particular method can be chosen according to a concrete application scenario.

Third, the size n of the set CS affects the configuration privacy. If n is small, $\mathcal{V}$ might have a high probability in guessing the configuration $cs_{\mathcal{P}}$. Therefore, to keep $cs_{\mathcal{P}}$ private, $\mathcal{P}$ should execute the protocol only if CS is of acceptable size. Moreover, $\mathcal{P}$ has to ensure that $\mathcal{V}$ cannot learn $cs_{\mathcal{P}}$ by running the PBA protocol multiple times with different configuration sets, because in the case of several successful attestations, $\mathcal{V}$ would know that $cs_{\mathcal{P}}$ is in the (possibly small) intersection of the sets used in the protocol executions. This example shows that $\mathcal{P}$ should install a privacy policy which prevents such abuses of the PBA protocol.

Fourth, note that the overhead of the TPM compared to binary attestation is small. Additionally, the TPM has to form the commitment C, which must be signed instead of $cs_{\mathcal{P}}$. So the overhead is just choosing a random number r and performing a modular multi-exponentiation modulo P (with two exponents). As with binary attestation, the TPM has to generate one digital signature (e.g., 2048 bit RSA). The TPM's computation does not depend on the size of CS.

6 Security of Our PBA Scheme

Here, we define a formal (game-based) security model based on the system model from Section 2, and state theorems about the security of our PBA scheme.

6.1 Security Model

Adversary model. The adversary $\mathcal{A}$ is a PPT algorithm and an active adversary that has full control over the communication channel between $\mathcal{H}$ and $\mathcal{V}$. This is modeled by the query of the form $\texttt{send}(E, m)$ which allows $\mathcal{A}$ to address a message m to an entity $E \in \{\mathcal{H}, \mathcal{V}\}$. In response, A receives a message which would be generated by E according to the protocol execution. In the definition of entity authentication, in which malicious hosts should also be considered, $\mathcal{A}$ is also given access to another query $\texttt{sendTPM}(m)$ by which it can communicate with $\mathcal{M}$. We assume that m contains the identity of the sender (as chosen by $\mathcal{A}$).

Moreover, when considering evidence authentication, the adversary may corrupt the host via the query $\texttt{corrupt}_{\mathcal{H}}$, which returns the configuration $cs_{\mathcal{P}}$ to $\mathcal{A}$ ($cs_{\mathcal{P}}$ is $\mathcal{H}$'s only secret).

We assume that $\mathcal{A}$ cannot corrupt the TPM. In reality, a hardware attack would be necessary to corrupt a TPM, i.e., we limit the adversary to software-only attacks, which is the assumption of the TCG [25]. In case a real-world adversary succeeds in attacking the TPM, our protocol has to rely on the revocation mechanisms for TPM signatures.

Evidence authentication. We formalize the intuitive security requirement that $\mathcal{A}$ should not be able to pretend that $\mathcal{P}$ has a configuration $cs_{\mathcal{P}}$ satisfying the property that has to be attested (i.e., $cs_{\mathcal{P}} \in CS$), when in fact the property is not fulfilled (i.e., $cs_{\mathcal{P}} \notin CS$).

Let $\mathsf{Game}_{\mathcal{A}}^{\text{ev-auth}}(1^\kappa)$ be the following interaction between $\mathcal{P}$, $\mathcal{V}$, and $\mathcal{A}$. Before the interaction, $\mathcal{A}$ chooses a platform with a valid TPM $\mathcal{M}$ and with a configuration $cs_{\mathcal{P}} \notin CS$. Then $\mathcal{A}$ is given access to $\texttt{send}(E, m)$, $\texttt{sendTPM}(m)$, and $\texttt{corrupt}_{\mathcal{H}}$ queries to any $\mathcal{P}$ chosen by $\mathcal{A}$. Uncorrupted parties behave as specified by the protocol. $\mathcal{A}$ wins, if it outputs a PBA signature σ, such that $\mathsf{PBA\text{-}Verify}$ accepts σ. We denote the success probability of $\mathcal{A}$ by $\mathsf{Succ}_{\mathcal{A}}^{\text{cf-priv}}(1^\kappa) := \Pr[\mathsf{Game}_{\mathcal{A}}^{\text{ev-auth}}(1^\kappa) = \mathsf{win}]$, and its maximum over all PPT adversaries $\mathcal{A}$ (running in time κ) as $\mathsf{Succ}^{\text{cf-priv}}(1^\kappa)$.

A PBA protocol provides evidence authentication if $\mathsf{Succ}^{\text{cf-priv}}(1^\kappa)$ is negligible in κ.

Configuration privacy. The security requirement that the configuration $cs_{\mathcal{P}}$ of $\mathcal{P}$ should be kept private is captured by the following game. For this requirement, host $\mathcal{H}$ and TPM $\mathcal{M}$ of $\mathcal{P}$ have to be honest because $\mathcal{P}$ could always send $cs_{\mathcal{P}}$ to $\mathcal{A}$.

Let $\mathsf{Game}_{\mathcal{A}}^{\text{cf-priv}}(1^\kappa)$ be the following interaction between $\mathcal{P}$, $\mathcal{V}$ and $\mathcal{A}$. $\mathcal{A}$ is given access to $\texttt{send}(E, m)$ queries. Moreover, $\mathcal{A}$ may access $\texttt{sendTPM}(m)$ and $\texttt{corrupt}_{\mathcal{H}}$ queries for all but one prover $\mathcal{P}$ chosen adaptively by $\mathcal{A}$, which has to remain honest. At the end of the interaction, $\mathcal{A}$ outputs an index i. $\mathcal{A}$ wins if i is the index of $\mathcal{P}$'s configuration in the set $CS = \{cs_1, \ldots, cs_n\}$, i.e., if $cs_{\mathcal{P}} = cs_i$. We denote the advantage of $\mathcal{A}$ (over a random guess) with $\mathsf{Adv}_{\mathcal{A}}^{\text{cf-priv}}(1^\kappa, n) := |\Pr[\mathsf{Game}_{\mathcal{A}}^{\text{cf-priv}}(1^\kappa) = \mathsf{win}] - 1/n|$, and its maximum over all PPT adversaries $\mathcal{A}$ (running in time κ) as $\mathsf{Adv}^{\text{cf-priv}}(1^\kappa, n)$.

A PBA protocol provides configuration privacy if $\mathsf{Adv}^{\text{cf-priv}}(1^\kappa, n)$ is negligible in κ.

Security of PBA. A PBA scheme is *secure*, if and only if it provides both evidence authentication and configuration privacy.

Trust relations. The TPM is assumed to be trusted by both host and verifier. For evidence authentication, a PBA protocol must ensure that a malicious host cannot cheat an honest verifier, whereas for configuration privacy, it must prevent a verifier controlled by $\mathcal{A}$ from determining the configuration of an honest host.

6.2 Security Analysis

The following theorems demonstrate the security of our PBA scheme. For the proofs, see Appendix A.

Theorem 1 (Evidence Authentication). *The PBA protocol presented in Section 5 provides evidence authentication (in the random oracle model), assuming the security of the ring signature scheme, the security of TPM signatures, and the hardness of the discrete logarithm assumption. In more detail:*

$$\mathsf{Succ}^{\textit{cf-priv}}(1^\kappa) \quad \leq \quad q^2/2^{\ell_\varnothing} + \varepsilon_{\mathit{TPM}} + \varepsilon_{\mathit{ring}} + \varepsilon_{\mathit{dlog}},$$

where q is the number of protocol runs, $\ell_\varnothing$ is polynomial in the security parameter κ, $\varepsilon_{\mathit{TPM}}$ is the probability of an adversary to forge a TPM signature, $\varepsilon_{\mathit{ring}}$ is the probability to forge a ring signature, and $\varepsilon_{\mathit{dlog}}$ is the probability to solve the underlying discrete logarithm problem.

Remark. Our proof does not directly use the random oracle model, however, it is required by the ring signature scheme we use.

Theorem 2 (Configuration Privacy). *The PBA protocol presented in Section 5 provides configuration privacy against computationally unbounded adversaries, due to the unconditional signer anonymity of the ring signature scheme and perfect hiding of the commitment scheme.*

Remark. Although our definition of configuration privacy assumes a PPT adversary (which would be reasonable for practical purposes), our protocol offers even unconditional security, because we use a perfectly hiding commitment scheme and an unconditionally signer-anonymous ring signature scheme.

7 Conclusion and Future Work

The concept of property-based attestation (PBA) has been proposed to overcome several deficiencies of the (binary) attestation scheme proposed by the Trusted Computing Group (TCG). Amongst others, the TCG attestation reveals the system configuration to third parties that could misuse it for privacy violations and product discrimination.

In this paper, we proposed the first cryptographic protocol for PBA which, in contrast to the previous solutions, does not require a Trusted Third Party to certify properties. In our protocol, the TPM has to compute only one commitment and one signature.

Furthermore, the cryptographic technique used here might be of independent interest: We demonstrate how a ring signature can be employed to prove membership in a list.

Future work may include the investigation of how to determine meaningful properties. Moreover, a generic approach based on any ring signature scheme, an efficient scheme with a security proof in the standard model, and the design of a PBA protocol with sub-linear communication and computation complexity in the size of the configuration set CS are still open problems.

References

1. Abe, M., Ohkubo, M., Suzuki, K.: 1-out-of-n signatures from a variety of keys. In *ASIACRYPT 2002*, LNCS vol. In: Zheng, Y. (ed.) ASIACRYPT 2002. LNCS, vol. 2501, pp. 415–432. Springer, Heidelberg (2002)
2. Bender, A., Katz, J., Morselli, R.: Ring Signatures: Stronger Definitions, and Constructions without Random Oracles. In: Halevi, S., Rabin, T. (eds.) TCC 2006. LNCS, vol. 3876, pp. 60–79. Springer, Heidelberg (2006)
3. Brickell, E., Camenisch, J., Chen, L.: Direct anonymous attestation. In: Pfitzmann, B., Liu, P. (eds.) Proceedings of ACM CCS 2004, pp. 132–145. ACM Press, New York (2004)
4. Brickell, E., Chen, L., Li, J.: A new direct anonymous attestation scheme from bilinear maps. In: Lipp, P., Sadeghi, A.-R., Koch, K.-M. (eds.) TRUST 2008. LNCS, vol. 4968. Springer, Heidelberg (2008)
5. Brickell, E., Li, J.: Enhanced Privacy ID: A direct anonymous attestation scheme with enhanced revocation capabilities. In: Proceedings of the 6th Workshop on Privacy in the Electronic Society (WPES 2007), pp. 21–30. ACM Press, New York (2007)
6. Camenisch, J.: Better privacy for trusted computing platforms. In: Samarati, P., Ryan, P.Y.A., Gollmann, D., Molva, R. (eds.) ESORICS 2004. LNCS, vol. 3193, pp. 73–88. Springer, Heidelberg (2004)
7. Camenisch, J., Michels, M.: Proving in Zero-Knowledge that a Number Is the Product of Two Safe Primes. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 107–122. Springer, Heidelberg (1999)
8. Camenisch, J., Stadler, M.: Proof Systems for General Statements about Discrete Logarithms. Technical Report TR 260, Dep. of Computer Science, ETH Zürich (March 1997)
9. Chandran, N., Groth, J., Sahai, A.: Ring Signatures of Sub-linear Size Without Random Oracles. In: Arge, L., Cachin, C., Jurdziński, T., Tarlecki, A. (eds.) ICALP 2007. LNCS, vol. 4596, pp. 423–434. Springer, Heidelberg (2007)
10. Chaum, D., van Antwerpen, H.: Undeniable signatures. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 212–216. Springer, Heidelberg (1990)
11. Chen, L., Landfermann, R., Löhr, H., Rohe, M., Sadeghi, A., Stüble, C.: A Protocol for Property-Based Attestation. In: Proceedings of ACM STC 2006, pp. 7–16. ACM Press, New York (2006)
12. Dodis, Y., Kiayias, A., Nicolosi, A., Shoup, V.: Anonymous identification in ad hoc groups. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 609–626. Springer, Heidelberg (2004)
13. Fujisaki, E., Okamoto, T.: Statistical Zero Knowledge Protocols to Prove Modular Polynomial Relations. In: Kaliski Jr., B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 16–30. Springer, Heidelberg (1997)
14. Kühn, U., Selhorst, M., Stüble, C.: Realizing Property-Based Attestation and Sealing on Commonly Available Hard- and Software. In: ACM STC 2007, pp. 50–57. ACM Press, New York (2007)
15. Micali, S., Rabin, M.O., Kilian, J.: Zero-Knowledge Sets. In: Proceedings of the 44th Symposium on Foundations of Computer Science (FOCS 2003), pp. 80–91. IEEE Computer Society, Los Alamitos (2003)
16. National Institute of Standards and Technology (NIST). Secure Hash Standard (SHS). FIPS PUB 180-2 (August 2002)

17. National Institute of Standards and Technology (NIST). Digital Signature Standard (DSS). FIPS PUB 186-3 (Draft) (March 2006)
18. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 129–140. Springer, Heidelberg (1992)
19. Poritz, J., Schunter, M., van Herreweghen, E., Waidner, M.: Property Attestation – Scalable and Privacy-friendly Security Assessment of Peer Computers. IBM Research Report RZ 3548 (# 99559) (October 2004)
20. Rivest, R., Shamir, A., Tauman, Y.: How to Leak a Secret. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248, pp. 552–565. Springer, Heidelberg (2001)
21. Sadeghi, A., Stüble, C.: Property-based attestation for computing platforms: Caring about properties, not mechanisms. In: Proceedings of NSPW 2004, pp. 67–77. ACM Press, New York (2004)
22. Schnorr, C.P.: Efficient Signature Generation by Smart Cards. J. Cryptology 4(3), 161–174 (1991)
23. Shacham, H., Waters, B.: Efficient Ring Signatures without Random Oracles. In: Okamoto, T., Wang, X. (eds.) PKC 2007. LNCS, vol. 4450, pp. 166–180. Springer, Heidelberg (2007)
24. Shoup, V.: Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint Archive, Report 2004/332 (2004), http://eprint.iacr.org/2004/332
25. Trusted Computing Group. TCG TPM Specification, Version 1.2, https://www.trustedcomputinggroup.org/

A Security Proofs

Proof (Evidence Authentication). We structure the proof as a sequence of games [24], where a PPT adversary $\mathcal{A}$ (see Section 2 for the adversary model) interacts with a simulator $\mathcal{S}$. The first game is $\mathsf{Game}_{\mathcal{A}}^{\text{ev-auth}}$. In each subsequent game, a new "event" is introduced. $\mathcal{S}$ aborts, whenever this event occurs. We show that each event can only happen with negligible probability for any PPT adversary, hence the probability for $\mathcal{A}$ to win game $\mathbf{G}_{i+1}$, denoted by $\Pr[\mathsf{win}_{i+1}]$, differs only by a negligible amount from its probability $\Pr[\mathsf{win}_i]$ to win game $\mathbf{G}_i$.

$\mathbf{G}_0$. The initial game is $\mathsf{Game}_{\mathcal{A}}^{\text{ev-auth}}$, where $\mathcal{S}$ plays the game with $\mathcal{A}$ by simulating the honest parties as specified by the protocol. $\mathcal{A}$ chooses a platform with a configuration $cs_{\mathcal{P}} \notin CS$ of his choice (as specified in Section 6.1), and $\mathcal{S}$ simulates the honest TPM $\mathcal{M}$ of this platform. $\mathcal{A}$ wins $\mathsf{Game}_{\mathcal{A}}^{\text{ev-auth}}$, and hence $\mathbf{G}_0$, if it manages to output $\sigma = (C, \sigma_{\mathcal{M}}, \sigma_r)$ such that $\mathcal{S}$ (acting as an honest verifier) accepts σ as a proof that $cs_{\mathcal{P}} \in CS$, although actually $cs_{\mathcal{P}} \notin CS$. Because $\mathbf{G}_0$ is $\mathsf{Game}_{\mathcal{A}}^{\text{ev-auth}}$, we have $\Pr[\mathsf{win}_0] = \mathsf{Succ}^{\text{cf-priv}}(1^\kappa)$.

$\mathbf{G}_1$. In the event that $\mathcal{S}$, acting as a verifier, chooses a nonce N_v that already occurred in a previous protocol run, $\mathcal{S}$ aborts the simulation. For this comparison, $\mathcal{S}$ records all nonces. As N_v is chosen randomly by $\mathcal{S}$, the probability ε_1 of this is $\leq q^2/2^{\ell_\varnothing}$ (which is negligible in the security parameter), where q denotes the number of protocol runs. Hence, $\mathsf{Succ}^{\text{cf-priv}}(1^\kappa) \leq \Pr[\mathsf{win}_1] + \varepsilon_1$.

$\mathbf{G}_2$. $\mathcal{S}$ simulates protocol execution as before, with the difference that all TPM signatures are obtained from the corresponding signing oracle. In the event that $\mathcal{S}$ receives an output $(C, \sigma_{\mathcal{M}}, \sigma_r)$ from $\mathcal{A}$, where $\sigma_{\mathcal{M}}$ was not created previously by $\mathcal{S}$, the simulation is aborted. In this case, $\mathcal{A}$ provided $\mathcal{S}$ with a forgery of a TPM signature. The probability $\varepsilon_{\mathsf{TPM}}$ of this event is the probability of a forgery of a TPM signature. Thus, $\mathsf{Succ}^{\text{cf-priv}}(1^\kappa) \leq \Pr[\mathsf{win}_2] + \varepsilon_1 + \varepsilon_{\mathsf{TPM}}$.

$\mathbf{G}_1$ covers replay attacks by estimating the probability that the same nonce occurs twice, and $\mathbf{G}_2$ covers forgeries of TPM signatures. It remains to estimate the probability $\Pr[\mathsf{win}_2]$. We consider two cases: either $\mathcal{A}$ wins in $\mathbf{G}_2$ by forging the ring signature (with probability $\varepsilon_{\mathsf{ring}}$), or without it. Since we are interested in the overall probability of $\mathcal{A}$ winning in $\mathbf{G}_2$, we do not require from $\mathcal{S}$ to detect which of these distinct cases occurs.

If no forgery of the ring signature occurred, but $\mathcal{A}$ wins $\mathbf{G}_2$, $\mathcal{A}$ must know a secret key r' matching one of the public keys used to compute the ring signature. Hence, $\mathcal{A}$ must know r', such that $h^{r'} = C/g^{cs_j} = g^{cs_{\mathcal{P}} - cs_j} h^r \bmod P$ for some $j \in \{1, \ldots, n\}$. Because $cs_{\mathcal{P}} \neq cs_j$, we have $r \neq r'$, thus $\mathcal{A}$ could compute the discrete logarithm $\log_g(h) = (cs_{\mathcal{P}} - cs_j)/(r' - r) \bmod Q$. The probability of the adversary to win the last game is $\Pr[\mathsf{win}_2] = \varepsilon_{\mathsf{ring}} + (1 - \varepsilon_{\mathsf{ring}}) \cdot \varepsilon_{\mathsf{dlog}} \leq \varepsilon_{\mathsf{ring}} + \varepsilon_{\mathsf{dlog}}$, where $\varepsilon_{\mathsf{dlog}}$ is the probability to solve the underlying discrete logarithm problem.

Thus, in total, $\mathsf{Succ}^{\text{cf-priv}}(1^\kappa) \leq \varepsilon_1 + \varepsilon_{\mathsf{TPM}} + \varepsilon_{\mathsf{ring}} + \varepsilon_{\mathsf{dlog}}$, which is negligible in κ if the TPM signature and ring signature schemes are secure and the underlying discrete logarithm problem is hard. □

Note that although our proof is in the standard model, the ring signature scheme in [1] requires the random oracle model.

Proof (Configuration Privacy). We demonstrate that $\mathsf{Adv}^{\text{cf-priv}}(1^\kappa, n)$, the maximum advantage over all $\mathcal{A}$ in $\mathsf{Game}^{\text{cf-priv}}_{\mathcal{A}}$, is negligible in κ, even if the adversary is computationally unbounded. For this, we construct a simulator $\mathcal{S}$ that plays $\mathsf{Game}^{\text{cf-priv}}_{\mathcal{A}}$ with some $\mathcal{A}$, simulating the honest parties. The goal of $\mathcal{A}$ is to break the configuration privacy of the PBA scheme, and the simulator's goal is to break either the perfect hiding property of the commitment scheme or the unconditional signer ambiguity property of the ring signature scheme.

We play the game twice. In the first case, we assume that the ring signature is secure and show how $\mathcal{S}$ can break the commitment scheme. In the second case, we assume that the commitment scheme is secure, and hence, we show how $\mathcal{S}$ can break the ring signature scheme.

Case 1. In this case, $\mathcal{S}$ is given a commitment $C = g^{cs_{\mathcal{P}}} \cdot h^r \bmod P$ with $cs_{\mathcal{P}} \in CS$, and plays $\mathsf{Game}^{\text{cf-priv}}_{\mathcal{A}}$ with $\mathcal{A}$.

Once $\mathcal{S}$ receives a `send` query with a nonce N_v from $\mathcal{A}$, it uses C in the PBA protocol execution as the TPM's commitment (without knowing $cs_{\mathcal{P}}$ and r), and creates a TPM signature $\sigma_{\mathcal{M}} = \mathtt{SignM}(sk_{\mathcal{M}}; (C, N_v))$. The computationally unbounded simulator $\mathcal{S}$ can compute α, such that $h = g^\alpha \bmod P$, and k, such that $C = g^k = g^{cs_{\mathcal{P}} + \alpha r} \bmod P$. Although $\mathcal{S}$ knows neither $cs_{\mathcal{P}}$ nor r, it can

establish n equations $k = cs_j + \alpha \cdot r_j$ (for $j = 1, \ldots, n$). Thus, $\mathcal{S}$ can compute n pairs (cs_j, r_j), and create the ring signature $\sigma_r = \texttt{SigRing}(r_j; \{y_j\}; N_v)$, where $y_j = g^{\alpha r_j} = h^{r_j} \bmod P$, with any of these r_j as a signing key. Because of the signer ambiguity of the ring signature scheme, $\mathcal{S}$ can choose an arbitrary r_j (for $j \in_R \{1, \ldots, n\}$). $\mathcal{S}$ sends C, $\sigma_{\mathcal{M}}$, and σ_r to $\mathcal{A}$.

At the end of the game, $\mathcal{A}$ outputs an index i. $\mathcal{S}$ attacks the perfect hiding property of the commitment scheme by using the pairs (cs_j, r_j) computed above, and opening the commitment to (cs_i, r_i).

Because we assume that the ring signature is secure, the probability of $\mathcal{S}$ to break the commitment scheme successfully is the probability of $\mathcal{A}$ to determine i with $cs_i = cs_{\mathcal{P}}$. Thus, a non-negligible advantage $\mathsf{Adv}^{\mathsf{cf\text{-}priv}}$ implies that $\mathcal{S}$ can break the perfect hiding property.

Case 2. In this case, $\mathcal{S}$ is given public/private key pairs (y_j, x_j) $(j = 1, \ldots, n)$ for the ring signature scheme, and access to a signature oracle for ring signatures under this key ring. $\mathcal{S}$ can use the oracle to query ring signatures on arbitrary messages. The unconditional signer ambiguity states that $\mathcal{S}$ should not be able to find out which private key was used for signing (although $\mathcal{S}$ knows all public and private keys). $\mathcal{S}$ chooses $k \in_R \mathbb{Z}_Q$, and computes $cs_j = k - x_j \bmod Q$ for $j = 1, \ldots, n$. Then, $\mathcal{S}$ starts to play $\mathsf{Game}_{\mathcal{A}}^{\mathsf{cf\text{-}priv}}$ with $\mathcal{A}$.

Once $\mathcal{S}$ receives a `send` query with a nonce N_v from $\mathcal{A}$, it computes $C := g^k \bmod P$ and $\sigma_{\mathcal{M}} := \texttt{SignM}(sk_{\mathcal{M}}; (C, N_v))$. $\mathcal{S}$ uses the ring signature oracle to create a ring signature σ_r on the message N_v, and sends C, $\sigma_{\mathcal{M}}$, and σ_r to $\mathcal{A}$.

At the end of the game, $\mathcal{A}$ outputs an index i. Since the commitment C was chosen randomly, the only possibility of $\mathcal{A}$ to win $\mathsf{Game}_{\mathcal{A}}^{\mathsf{cf\text{-}priv}}$ is to break the signer ambiguity of the ring signature. $\mathcal{S}$ also outputs i to indicate that x_i was used to generate the signature, thus breaking the unconditional signer ambiguity of the ring signature scheme. □

The Reduced Address Space (RAS) for Application Memory Authentication

David Champagne, Reouven Elbaz, and Ruby B. Lee

Department of Electrical Engineering, Princeton University
Princeton, NJ 08544, USA
{dav,relbaz,rblee}@Princeton.edu

Abstract. Memory authentication is the ability to detect unauthorized modification of memory. Existing solutions for memory authentication are based on tree structures computed over either the Physical Address Space (*PAS tree*) or the Virtual Address Space (*VAS tree*). We show that the PAS tree is vulnerable to *branch splicing* attacks when providing memory authentication to an application running on a potentially compromised operating system. We also explain why the VAS tree generates initialization and memory overheads so large as to make it impractical, especially on 64-bit address spaces. To enable secure and efficient application memory authentication, we present a novel *Reduced Address Space* (RAS) containing only those pages that are useful to a protected application at any point in time. We introduce the *Tree Management Unit* (TMU) to manage the *RAS tree*, a dynamically expanding memory integrity tree computed over the RAS. The TMU is scalable, enabling tree schemes to scale up to cover 64-bit address spaces. It dramatically reduces the overheads of application memory authentication without weakening the security properties or degrading runtime performance. For SPEC 2000 benchmarks, the TMU speeds up tree initialization and reduces memory overheads by three orders of magnitude on average.

Keywords: Memory authentication, integrity tree, secure computing architecture.

1 Introduction

As security-critical applications become mainstream, several research efforts [9, 12, 13, 16, 17] aim to design general-purpose computing platforms that can prevent physical and software attacks. A crucial security objective shared by these platforms is to protect the integrity of a sensitive application by providing it with *memory authentication*—i.e. the ability to verify that a data block it reads at a given address from main memory contains the data it last wrote at this address. To enable memory authentication without having to keep on-chip a fingerprint for each data block to protect, [1, 3, 4, 6, 14, 15, 18] propose constructing integrity trees.

An integrity tree is built by recursively computing cryptographic fingerprints on the protected memory blocks until a single fingerprint of the entire memory space, the tree root, is obtained. It can be computed over either the physical address space (PAS tree)

T.-C. Wu et al. (Eds.): ISC 2008, LNCS 5222, pp. 47–63, 2008.

or the virtual address space (VAS tree). In any case, the tree root is kept on-chip, in a tamper-resistant area, while the rest of the tree can be stored off-chip. Authenticating a block read from main memory requires fetching and verifying the integrity of all fingerprints on the tree branch starting at the block of interest and ending with the root.

Modern operating systems typically cannot be trusted to enforce such application memory space integrity, as exploitable software vulnerabilities are commonly found in these large, complex and extendable software systems. Hardware-driven integrity tree systems must thus be able to provide memory authentication to an application despite potential OS compromise. However, the majority of past research efforts [3, 4, 9, 15, 16, 18] build PAS trees, which are not secure in this setting: we show that by providing incorrect virtual-to-physical address translations, a compromised or malicious OS can trick the on-chip authentication engine of a PAS tree into fetching the wrong tree branch to verify a memory block. As a result, a memory block at a given address is passed as valid for a different address; we call this a *branch splicing* attack.

Application memory authentication despite a potentially compromised OS was described in [17], where a VAS tree is built. In this case, the on-chip authentication engine selects the tree branch to fetch using the virtual address of the block to verify, so a malicious OS cannot wage a branch splicing attack. However, tree node addressing requires that the VAS tree span a contiguous segment of memory. We show that as a result, the VAS tree must cover all memory located between the lowest and highest virtual addresses that may be accessed by the application during its execution. Such a large tree span leads to enormous initialization and memory overheads, making the deployment of the VAS tree impractical.

In this paper, we introduce a novel address space, the Reduced Address Space (RAS) containing only the application's memory footprint. We build an integrity tree over the RAS (a RAS tree) which remedies the security and efficiency problems of the PAS tree and VAS tree. Our RAS tree expands dynamically as the underlying RAS grows to fit the memory needs of the application.

The contributions of this paper are as follows:

- showing that a tree over a virtual address space (VAS tree) is impractical, while a tree over the physical address space (PAS tree) is insecure when the OS is untrusted;
- proposing a novel Reduced Address Space (RAS), where the pages used by an application form a contiguous region that expands dynamically to fit the application's memory needs;
- introducing the concept of a dynamic integrity tree to protect the pages in the RAS;
- detailing the design of a Tree Management Unit (TMU) which constructs as well as maintains the RAS and the dynamic integrity tree covering the RAS;
- reducing by three orders of magnitude the VAS tree overheads in storage and tree initialization, for a 4 GB memory space, without any significant impact on runtime performance.

The rest of the paper is organized as follows. Section 2 presents our security model. Section 3 presents the PAS and VAS trees, explains the branch splicing attack on the PAS tree and the enormous VAS tree overheads. Section 4 presents our RAS tree covering the novel Reduced Address Space and implemented by our Tree

Management Unit. Sections 5 and 6 analyze the security and performance of the approach we propose. Section 7 concludes the paper.

2 Security Model

Threat Model. Our threat model encompasses that of existing integrity trees, which aim to prevent the following attacks. *Spoofing attacks*: the adversary substitutes a fake block for an existing memory block. *Splicing or relocation attacks*: the attacker swaps a memory block at address *A* with a block at address *B*, where $A \neq B$. Such an attack may be viewed as a spatial permutation of memory blocks. *Replay attacks*: a memory block located at a given address is recorded and inserted at the same address at a later point in time; by doing so, the current block's value is replaced by an older one. Contemporary operating systems such as Windows and Linux consist of millions of lines of codes [5, 7]. Since the number of exploitable software defects per thousand lines of code has been evaluated to be between 3 and 6 [11], it is extremely difficult to assume an infallible operating system. The attacks described above can thus be carried out through a compromised OS (software attacks) or directly on the processor-memory bus, via probing and injection of fake values (physical attacks).

Trust Model. The main hypothesis of our trust model is that the processor chip itself is trusted, i.e., it is considered resistant to physical and side-channel attacks. For simplicity, this paper assumes the CPU contains a mechanism—like those of [12, 13, 17]—protecting the integrity of a sensitive application's on-chip state (registers and cache lines) on context switches. Similarly, we assume the CPU contains an on-chip engine to authenticate, upon launch, the on-disk image of a protected application.

3 Related Work

Upon reading or writing a memory block, a CPU maintaining a memory integrity tree must locate and fetch the tree nodes along the block's branch so tree verification or update procedures can take place (see Appendix A for background on integrity trees). In Section 3.1, we describe the methodology applied by computing platforms in the literature to perform this tree traversal. Section 3.2 and 3.3 then show how tree traversal affects the security and efficiency of the PAS tree and VAS tree proposed in past research.

3.1 Tree Traversal

[4] introduces a tree traversal methodology, adopted by [3], which derives the addresses of nodes on a block's branch from the block's address. The technique works as follows: starting from the tree root and going down towards the leaves, all tree nodes (including the leaves themselves) are assigned a numerical position (the P_x in Figure 1). The root is assigned position -1 (P_{-1}) and the other tree nodes are at position 0 (P_0) and higher.

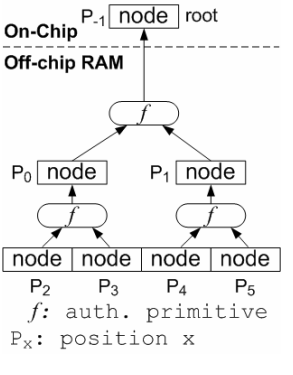


Fig. 1. A binary integrity tree

All tree nodes are of identical size and are laid out in memory according to their positions, starting with P_0 at address 0, followed by P_1, P_2, etc at higher addresses. With contiguous siblings, the node size and tree arity can be set to allow the on-chip authentication engine to fetch a node and its siblings (necessary to recompute the parent) in a single memory transaction using the node's address. EQ_1 is used to map the address (node_address) of a node N to a position (node_position). The function in EQ_2 then allows mapping N's position to the position of its parent (parent_position). EQ_2 can be applied recursively to obtain the position of nodes on N's branch. A parent_position equal to -1 indicates the root node has been reached. The address of any node other than the root can be obtained from its position using EQ_3.

$$node\_position = node\_address \div node\_size \quad (EQ_1)$$

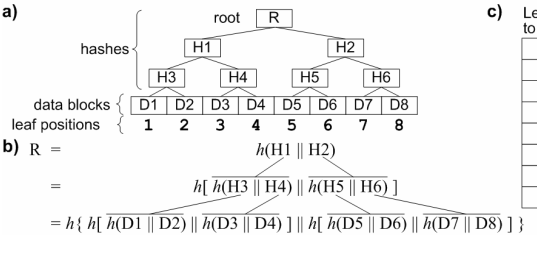
$$parent\_position = \lfloor node\_position \div arity \rfloor - 1 \quad (EQ_2)$$

$$node\_address = node\_position \times node\_size \quad (EQ_3)$$

This technique for generating node addresses is quick and efficient since it does not use indirection to go from one parent to another and it can be implemented in hardware at a low cost. However, it requires that all leaf nodes be part of a contiguous memory segment, i.e. the tree must cover a monolithic region of address space. This memory segment may be contiguous either in the physical address space (PAS tree, Section 3.2) or the virtual address space (VAS tree, Section 3.3).

3.2 Physical Address Space Tree (PAS Tree)

The majority of past research efforts [3, 4, 9, 15, 16, 18] considers the operating system as trusted and builds the integrity tree over the physical address space (a PAS tree). However, when the OS is not considered as a trusted system, as is the case in this paper, the PAS tree is vulnerable to attacks that can violate the integrity of the protected application's memory space. We show below that by corrupting the application's page table, a compromised or malicious OS can trick the on-chip authentication engine into using the wrong branch when verifying the integrity of a block. As a result, the PAS tree cannot provide application memory authentication with an untrusted OS.



c)

Leaf Position to Verify	Root Re-computation Performed by Authentication Engine*
1	*COMP1* = $h\{ h[h(D1 \| D2) \| H4] \| H2 \}$
2	*COMP2* = $h\{ h[h(D1 \| D2) \| H4] \| H2 \}$
3	*COMP3* = $h\{ h[H3 \| h(D3 \| D4)] \| H2 \}$
4	*COMP4* = $h\{ h[H3 \| h(D3 \| D4)] \| H2 \}$
5	*COMP5* = $h\{ H1 \| h[h(D5 \| D6) \| H6] \}$
6	*COMP6* = $h\{ H1 \| h[h(D5 \| D6) \| H6] \}$
7	*COMP7* = $h\{ H1 \| h[H5 \| h(D7 \| D8)] \}$
8	*COMP8* = $h\{ H1 \| h[H5 \| h(D7 \| D8)] \}$

Dx = Value of leaf data block to validate

Fig. 2. Binary Merkle Tree (a) with root value derivation (b) and root re-computation equations (c)

The Branch Splicing Attack. Figure 2a shows a binary Merkle tree covering 8 data blocks (labeled *D1* to *D8*) with a root *R* and intermediate hashes *H1* to *H6*. The value of *R* is derived in Figure 2b, where we see that the root and intermediate tree nodes are computed using the value of their respective descendents. Figure 2c shows how the on-chip authentication engine recomputes the root *R* when verifying a given leaf block; *COMP1* to *COMP8* use different leaves and intermediate hashes but when the application is not under attack, the result of all computations is the value of the root *R*.

In Figure 3a, we show how a malicious OS can corrupt a protected application's page table to control the virtual-to-physical address translations in the Translation Lookaside Buffer (TLB) such that the wrong tree branch is fetched and the wrong equation is used to recompute the root. For simplicity, we do not consider caching here. The upper path with solid lines represents the verification procedure when the system is not under attack, while the lower path with dotted lines shows the effect of an attack (with corrupted data items in dark grey). In its attack, the malicious OS successfully tricks the processor into accepting D7 rather than the correct D1 by forcing the authentication engine to fetch D7's branch and compute *COMP7* instead of *COMP1*. The OS does so by corrupting the application's page table—mirrored in the processor's TLB—so that it maps virtual address V(D1) to physical address P(D7). This causes the authentication engine to compute an erroneous leaf position, hence generating the wrong branch node addresses and using the wrong *COMPi*. We call this attack branch splicing, which successfully substitutes D7 for D1 without detection.

3.3 Virtual Address Space Tree (VAS Tree)

To provide application memory authentication despite an untrusted OS, [17] builds the integrity tree over the virtual address space (VAS tree). Figure 3b shows that with

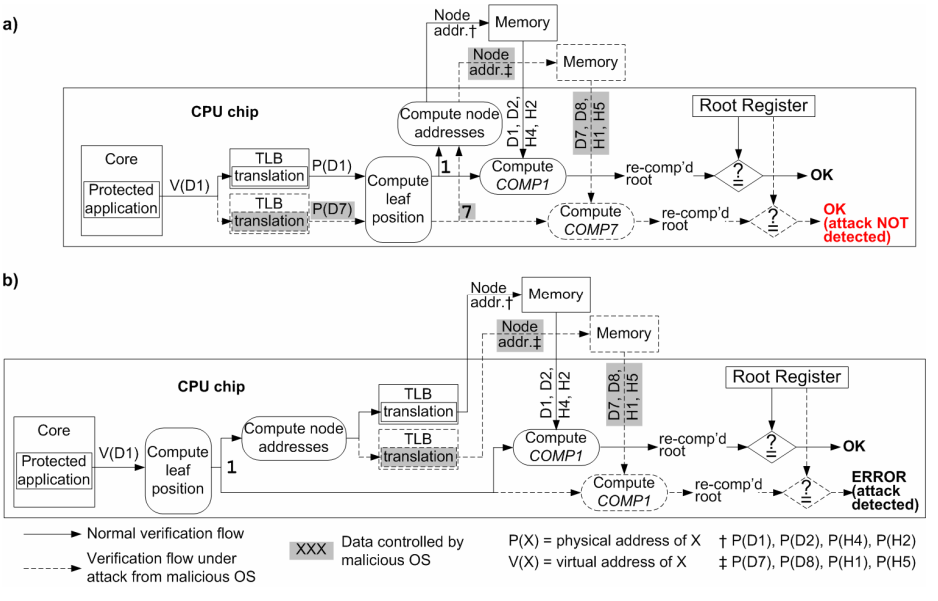


Fig. 3. Successful branch splicing attack on PAS tree (a) and unsuccessful attack on VAS tree (b)

a VAS tree, the OS cannot influence the choice of the *COMPi* equation used to recompute the root. The leaf position of a block is determined by that block's virtual address—which comes directly from the protected application—rather than by its translated physical address—which is under OS control. As a result, a branch splicing attack by the OS is detected by the authentication engine.

However, the extra security afforded by the VAS tree over the PAS tree comes at the cost of very large memory capacity and initialization overheads. Application code and data segments are usually laid out very far apart from one another in order to avoid having dynamically growing segments (e.g., the heap and stack) overwrite other application segments. The VAS tree must thus span a very large fraction of the virtual address space in order to cover both the lowest and highest virtual addresses that may be accessed by the application during its execution. The span of the tree is then several orders of magnitude larger than the cumulative sum of all code and data segments that require protection—i.e. the tree must cover vast regions of unused memory. In the case of a VAS tree protecting a 64-bit address space, the span can be so enormous as to make the VAS tree impractical, i.e. the VAS tree is not scalable. Indeed, it not only requires allocating physical page frames for the 2^{64} bytes of leaf nodes that are defined during initialization, but also requires allocating memory for the non-leaf tree nodes, which represent 20% to 100% of the leaf space size [3, 6, 17]. The CPU time required to initialize such a tree is clearly unacceptable in practice.

To overcome the enormous overheads of the VAS tree and defend against branch splicing attacks, we build the integrity tree over a new Reduced Address Space (RAS) managed by our TMU architecture.

4 The TMU Architecture

The role of the Tree Management Unit (TMU), a new processor hardware unit, is to maintain an integrity tree over the Reduced Address Space (RAS). At any point in time, the RAS contains only those pages needed for the application's execution; it grows dynamically as this application memory footprint increases. The TMU builds an integrity tree over the application's RAS (a *RAS tree*), carries out the integrity verification and tree update procedures and expands the tree as the underlying RAS grows. Because the RAS contains the application's memory footprint in a contiguous address space segment, the RAS tree does not suffer from the overheads of the VAS tree, built over a sparse address space. Moreover, the design of the TMU architecture ensures the RAS tree is not vulnerable to branch splicing.

4.1 Overview

Reduced Address Space (RAS). The approach presented in this paper consists in building and maintaining a dynamic integrity tree covering only the set of data and code pages required for the tamper-evident execution of an application. In order to do so, we construct a new address space, the RAS depicted in Figure 4, containing only this set of pages and descriptors for the memory regions whose pages are not in the RAS (the Unmapped Page Ranges, UPRs, Section 4.4). We then build the integrity tree over the RAS so the TMU can authenticate the application's code and data as

well as determine, by inspecting the tree-protected UPR list, whether a page is mapped in the RAS. Pages may be unmapped because 1) they haven't been used by the application yet, or 2) they are mapped to physical pages shared with other applications, or 3) they belong to the operating system. The novelty in our approach is that the integrity tree does cover the whole virtual address space, except that the unused memory regions are condensed—represented by just their first and last addresses in a UPR list item.

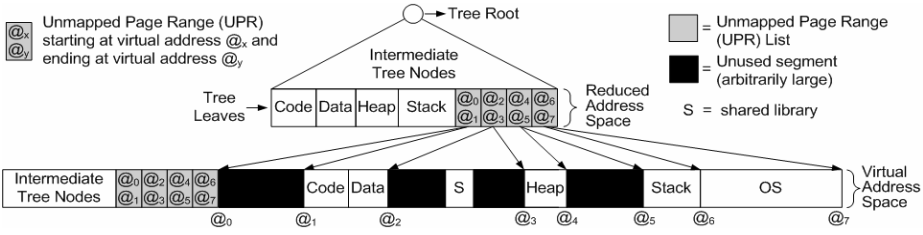


Fig. 4. The Reduced Address Space (RAS) and its relationship to the Virtual Address Space

The Tree Management Unit (TMU). Figure 5 depicts in dark gray the components added to a general-purpose processor to support our scheme. The role of the main addition, the TMU, is to initialize and expand the tree, as well as to traverse it from a leaf to the root on verifications and updates. The *Check&Update* logic is a crucial component: it retrieves the application image through the Authentication Engine to build the initial tree (Section 4.2). It also drives the *Page Initialization* logic on tree expansions (Section 4.3). The *RAS owner* register identifies the currently protected application; the TMU only operates when the RAS owner is in control of the CPU. The *RAS Ctr* gives new RAS indices. The *TMU tags* are used by the TMU to map a virtual page into the RAS while the node TLB (*N-TLB*) computes the physical address of a tree node.

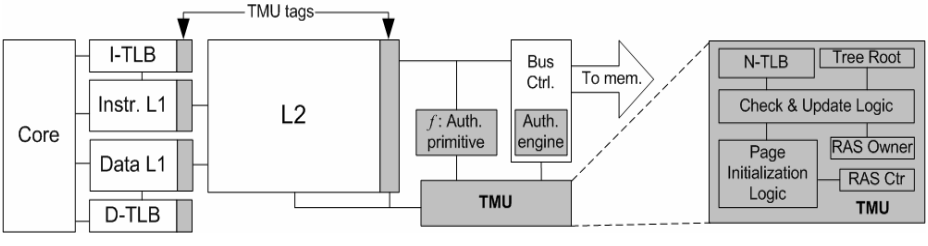


Fig. 5. The Tree Management Unit (TMU) architecture

4.2 Dynamic Integrity Tree over the Reduced Address Space (RAS)

The construction of the RAS follows the execution of the protected application. Initially empty, the RAS is populated by the TMU—when the protected application is launched—with the pages whose contents are defined in the application image.

During execution of the program, the TMU adds to the RAS pages that are accessed by the application and were not previously included in the RAS, e.g. a heap or stack page touched for the first time. The TMU adds a page to the RAS by assigning it an identifier—called the *RAS index*—generated from an on-chip counter (RAS Ctr in Figure 5), which is initially 0 and incremented for each new RAS page. The address of a datum within the RAS is *addr* = *RAS_index* || *offset*, where offset is the data's offset within its virtual page and || indicates concatenation.

Initialization of the RAS. Initializing a protected application being launched consists of building an integrity tree over a Reduced Address Space containing the code and data specified in the application image on the disk (e.g. an ELF executable file). We assume the image consists of a set of data and code segments, along with a header describing how these segments must be laid out by the loader to construct the application's initial virtual address space. In the scheme we propose, one of the data segments must be a UPR list specifying the memory regions that should not be mapped in the RAS by the TMU during initialization. Since we let the untrusted OS and its loader take the image from disk to memory, we must first authenticate the image before it is mapped into the RAS and included in the initial tree. We assume the CPU has an on-chip engine—similar to that in [10] or [13]—in charge of authenticating the application image. During initialization, the TMU reads via this engine the defined pages of the application's initial state and maps them into the RAS.

Upon mapping the first page into the RAS, the TMU sets RAS ownership by writing the RAS owner register with a value identifying the application. Whenever the RAS owner gains control of the CPU, TMU protection is activated; when the owner is pre-empted, the TMU is deactivated. The value identifying the application depends on what application-related information the ISA (Instruction Set Architecture) makes available to the hardware: it could be the process ID of the protected application or the base address of its page table. The RAS owner value is bound to the contents of the RAS: if a malicious OS assigns the RAS owner process ID to an application other than the actual owner, the TMU detects the subterfuge when the non-owner tries to execute an instruction that does not exist in the RAS.

4.3 Maintaining a Dynamic Integrity Tree over an Expanding RAS

The size of the RAS increases at a page granularity. As a result, our integrity tree must span an extra page every time a new page is mapped into the RAS. Such tree expansions trigger increases in the tree's span (number of leaves). We first define new concepts needed to describe runtime tree expansion.

Definitions. The term integrity tree refers to an A-ary tree structure used to provide memory authentication on a computing platform. In this paper, we consider that integrity trees are built as depicted in Figure 6, with the root node at the topmost level and the leaf nodes—which are the application memory blocks to authenticate—at the lowest level. We call intermediate nodes the nodes between level L=0 and the root; these nodes are metadata involved in the integrity checking process. All integrity trees are complete: they have A^{Lr} leaves, where Lr is the level to which the root belongs.

Fig. 6. A partial integrity tree covering 10 memory blocks (nodes 15 to 24). It can cover 6 extra blocks, mapped into nodes 25 to 30, before becoming a full tree.

The branch of a node *F* is the set of nodes on the shortest path from *F* to the root; this branch is the authentication path for *F*. The nodes on *F*'s branch are called *F*'s ancestors (*F*'s parent, the parent of *F*'s parent, etc). For example, the branch for node 17 in Figure 6 consists of nodes 8, 3, 1 and 0.

The minimum number of levels necessary to authenticate *Nb* memory blocks is $Lb = \log_A(Nb)$. The number of leaves *Nl* in the integrity tree is equal to A^{Lb}. Building the *Nl* branches of the integrity tree during the tree's initialization would be inefficient since the *Nb* blocks only required *Nb* distinct authentication paths, i.e. *Nb* tree branches. This paper presents a strategy to avoid computing and storing branches for the *Nl* - *Nb* undefined leaf nodes that do not correspond to one of the *Nb* memory blocks to authenticate. This strategy uses the notion of ghost nodes, the nodes which are not on the branches of the *Nb* defined memory blocks. In opposition to a ghost node, a solid node is a node on the branch of a defined memory block. The term nil node refers to a ghost node which is the sibling of a solid node. Only solid nodes need to be computed and stored in memory: ghost nodes are neither computed nor stored anywhere and thus do not cause CPU time or memory overheads. A nil node is fixed to a pre-determined value whenever used in the computation of its parent node.

A *full tree* is an integrity tree without ghost nodes. A *partial tree* has both solid and ghost nodes. A partial tree *T* can protect an additional memory block by replacing a ghost leaf node with a solid node, and then computing the solid node's ancestors. This can be done until *T* becomes a full tree.

For an integrity tree *T*, we define a cone to be a sub-tree of *T*, with a node from *T* as its root and nodes from *T*'s level 0 as its leaves. In Figure 6, nodes 4, 9, 10, 19, 20, 21 and 22 are part of a cone which has 4 as its root. A *page cone* is any cone whose leaves cover the contents of exactly one page. A *page node* is the root of such a cone. The tree has a level consisting exclusively of page nodes; it is called the page node level. This allows tree expansions at a page granularity and, as we explain next, ensures that the integrity tree (partial or full) always covers an integer number of pages. At any point in time, the memory blocks forming the pages in the RAS are the only solid leaf nodes in the integrity tree. The TMU manages the tree such that these nodes are always the leftmost nodes on level 0, i.e. the TMU expands the tree to the right.

Tree Expansion. To add a first page P_a to the empty tree T_0, the TMU assigns RAS index 0 (*RAS_i*=0) to P_a, computes its page cone and makes its page node R_1 the root of T_1, the new integrity tree (Figure 7 depicts a binary tree, with T_1 in 7a). When a second page P_b requires protection, T_1 must be expanded so its root spans both P_a and P_b. To do so, the TMU first computes P_b's page cone and then assigns it a RAS index equal to 1. Since T_1 is a full tree—i.e. it is formed only of solid nodes—the TMU must increase T_1's height by one, creating tree T_2 covering both P_a and P_b. The TMU computes T_2's root R_2, located on the new topmost level of the tree, by applying the

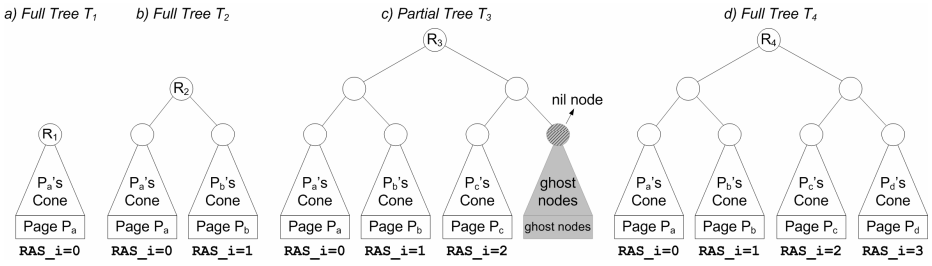


Fig. 7. Tree expansion: each page added expands the tree span. When adding a page to a full tree (a)(b), an extra tree level is added. When a page is added to a partial tree (c), it replaces a cone of ghost nodes.

authentication primitive (*f* in Figure 1) to the page nodes of P_a, P_b and their siblings. Generally, there are two different scenarios for tree expansion: i) a page needs to be added to a full tree or ii) a page needs to be added to a partial tree.

Expansion of a Full Tree. Upon adding a page P_n to a tree, the TMU can detect the integrity tree is full when the RAS index `RAS_i` to be assigned is a power of the arity, i.e. whenever $\log_A(\texttt{RAS\_i}) = x$, such that x is an integer larger than or equal to zero (e.g., in Figure 7b). Upon adding P_c, the TMU must thus increase the height of T_2 by one. To do so, the TMU creates a new level containing a single solid node R_3 on top of the existing tree, computes P_c's cone and then computes a branch starting at P_c's page node and ending at R_3. In computing this branch, the ghost nodes which are children of an ancestor of P_c are nil nodes. The value attributed to nil nodes for the computation of their parents depends on the underlying integrity tree e.g., for a hash tree, a nil node is a null value. The last step of the branch computation procedure uses the old tree root R_2 to make R_3 into the new tree root. Finally, the on-chip R_2 is replaced by R_3: the former becomes a regular intermediate node and can be stored in external memory.

Expansion of a Partial Tree. When the integrity tree is partial (e.g., Figure 7c), a page P_n to be added is assigned the next available RAS index and its page cone is computed (e.g., Figure 7d). The leftmost ghost node in the page node level of the partial tree is then replaced with P_n's page node. The ancestors of P_n's page node are then recomputed so the root reflects the expanded RAS. When the page node level is filled with solid nodes, we have a full tree.

In this paper, we consider that tree expansions only occur to integrate new heap or stack pages into the tree. Since programs do not expect any specific values to be present in a newly allocated stack or heap page, the contents of such pages can be forced to all zeroes without any impact on program correctness. Thus the TMU does not need to fetch the contents of a page upon a tree expansion: it merely initializes a stack or heap page by writing zeros directly on the memory bus and computes its page cone concurrently.

4.4 UPR List and TMU Tags

In order to maintain the tree and address its nodes, the TMU uses the tags depicted in Figure 5, which are composed of three fields each: the `TMU_field`, the `mapped` bit

(M) and the `excluded` bit (E). These fields are stored in the OS memory space, in extensions of the Page Table Entries (PTEs). They are carried on-chip into extensions of the Translation Lookaside Buffer (TLB) entries on TLB misses, propagated to cache lines on a cache line fill and made available to the TMU on a cache line miss or eviction.

The `mapped` bit tells the TMU whether a page is currently assigned a valid RAS index. When the `mapped` bit is set, the `TMU_field` contains the page's RAS index; otherwise, it contains an index pointing to the UPR list entry confirming that the page is indeed part of an unmapped region. The application's `excluded` bits are specified in its image by the application creator and do not change during execution. The TMU does not map into the RAS pages with an `excluded` bit set to 1. These pages are thus excluded from the tree's coverage; the application can read and write them without triggering tree verification or updates. This is necessary to allow the OS and other applications to share data with the protected application. As in [17], we store a virtual tag with each physically-tagged cache line to be able to compute parent node addresses when the line is evicted. TMU tags thus contain a fourth field for physically-tagged caches.

UPR List. The UPR list (UPRL) is a list of Unmapped Page Ranges (UPRs). A UPR consists of 1) a pair of virtual page numbers defining a range of contiguous virtual pages not currently in the RAS and 2) an excluded bit specifying whether or not the pages in the range should be excluded from tree coverage. The UPRs in a UPRL are mutually exclusive and the sum of all UPRs spans all virtual pages that are not covered by the tree (Figure 4). We store the UPRL in the RAS so that its integrity is protected by the tree.

Upon adding a page P to the tree, the TMU must update the UPR list to remove P from its UPR. If P with address X is the first page within its UPR $U = X \parallel Y \parallel 0$ (where 0 is the `excluded` bit), the TMU simply overwrites U with X+1 $\parallel$ $Y \parallel 0$. Similarly, if Y is the last page of a UPR, then the TMU writes $X \parallel Y\text{-}1 \parallel 0$. If P is not one of the bounds of its UPR $U = X \parallel Y \parallel 0$ (where $X < P < Y$) however, removing P from the UPRL creates two ranges, $U_1 = X \parallel P - 1 \parallel 0$ and $U_2 = P$+1 $\parallel$ $Y \parallel 0$. To update the UPR list, the TMU changes U to U_1 and appends the newly created U_2 to the end of the UPRL. While this UPRL maintenance strategy is quick and efficient, it creates an unordered list in which searching can be a demanding operation.

To allow the TMU to access the UPR list in constant time without searching, we provide, in the `TMU_field` of an unmapped page P, an index—the `UPRLi`—pointing to the UPR containing P. When the physical address translation or the UPRL index of an unmapped page (M = 0) is undefined in a TLB entry, a page fault is raised and the OS page fault handler intervenes to provide the missing information. In order to be able to provide the appropriate UPRL index on a page fault, the OS needs to be aware of the exact contents of the UPRL. To do so, the OS can read the application's UPRL directly by mapping it in its own address space. The OS can also maintain, in a data structure with a good search complexity, a copy of the UPRL which it updates every time a page is added to the tree by the TMU. Note that the TMU detects a fake `UPRLi` provided by a malicious OS upon accessing the tree-protected UPR list.

5 Security Analysis

In this section, we argue the RAS tree and the TMU do not degrade the security properties of the underlying tree scheme (e.g., Merkle Tree). Analysis of tree traversal shows our RAS tree defends against branch splicing attacks. We begin our analysis with tree expansion, a key TMU mechanism.

Tree Expansion. Let T be a tree covering the X-page wide RAS R. Upon mapping a new page P into R to form R', the tree expansion procedure must transform T into a tree T' that can be used to authenticate R'. To do so, the TMU computes P's page cone C_P and integrates C_P into the tree. C_P's computation does not degrade the security properties of the underlying tree since 1) the integrity of the data used as leaves by the TMU in this step is verified and 2) the root of the intermediate trees formed during the construction of C_P are kept on-chip. When P is defined in the image, property 1) is guaranteed by the authentication engine's validation of P's data leaves; when P is a dynamically allocated stack or heap page, the data is fixed to the predetermined zero value so property 1) holds. Property 2) is by design.

Integration of C_P into T consists of updating the branch for the cone's root—i.e. C_P's branch—to make the root of T' reflect the state of R'. The integration of C_P into T preserves the security properties of the underlying integrity tree because 1) the addresses of the branch nodes to be updated are computed by the TMU from a trusted RAS index and 2) the nodes on the new branch are computed over verified data. Property 1) holds because the addresses of branch nodes are derived from the RAS index allocated to P, which is obtained from a trusted on-chip counter. Property 2) holds because the TMU computes the new branch by recursively applying the authentication primitive on trusted nodes—i.e. C_P's root and verified siblings (nil nodes or preexistent solid nodes that were verified prior to the cone's computation).

Initialization. The security of the initialization procedure follows from security of the tree expansion procedure. Indeed, initialization consists in recursive invocations of the tree expansion procedure, initially on the empty tree. The TMU figures out which pages of the virtual address space to add to the initial tree by reading the image header through the authentication engine. The procedure thus builds an integrity tree covering all the data and code defined in the image without degrading the security properties of the underlying tree.

Runtime RAS Expansion is guided by the excluded and mapped bits of pages touched by the application. The TMU checks the integrity of these bits by looking up the UPR list and thus avoids mapping into the RAS pages that are already in the RAS or are excluded from tree protection. Hence, the excluded and mapped bits ensure an attacker cannot force the RAS expansion procedure to map into the RAS a page that should not be mapped.

Tree traversal. With regard to tree traversal, the only difference between a VAS tree and our RAS tree lies in the indirection mechanism provided by the RAS to determine node addresses. When computing the addresses of the nodes on a datum's branch, we rely on both its RAS index and the datum's virtual address. We thus need to ensure they are genuine. The virtual addresses used by the application are genuine since they are generated by integrity-protected instructions operating over integrity-protected

data—i.e. both instructions and data are covered by the tree. Since each node in the tree is computed using the virtual address of its children as an input to *f*, and since the verification branch taken by the TMU depends on the RAS index used, the last step of the branch verification procedure—verification against the root—not only checks the integrity of a datum but also checks that the right RAS index was used to verify the datum. An OS trying to wage a branch splicing attack by corrupting the RAS index will thus be detected.

6 Performance Evaluation

We now evaluate the impact of the RAS tree on initialization (Section 6.1) and runtime overheads (Section 6.2). Our evaluation was carried out using the execution-driven, cycle-accurate SimpleScalar simulator [2] to run nine SPEC2000 benchmarks [8] as representative applications: `applu`, `art`, `gcc`, `gzip`, `mcf`, `swim`, `twolf`, `vortex` and `vpr`. These are typically used by the computer architecture community to evaluate performance of microprocessor features and implementations. Our RAS tree and the VAS tree we use as a base case are implemented using the cached Merkle hash tree scheme described in [4] (see Appendix A for background on cached integrity trees). The hardware architectural parameters used in our simulations are in Table 1. To simulate paging, we implemented a simple LRU page replacement algorithm and assumed the OS could service page faults infinitely fast (i.e. the OS is only limited by the throughput of the hard drive). Unless mentioned otherwise, the memory span of applications is 32-bit, i.e. the distance between the start address of the lowest application segment and the end address of the highest application segment is 2^{32} bytes. The benefit of our RAS approach will increase exponentially as the address space grows to 64 bits.

6.1 Initialization

We first show that dynamic trees built over the RAS and managed by the TMU reduce the memory capacity and CPU time by several orders of magnitude over equivalent VAS trees. *Note that the Y-axis of all figures in this section is in logarithmic scale.* Figure 8 compares the memory overheads caused by VAS and RAS trees during initialization. RAS trees have an initialization-time memory overhead which varies from one application to another since they are initialized to cover only those pages with contents defined by the image. `applu` and `swim` have the largest amount of

Table 1. Architectural parameters

Parameter	Value
Clock frequency	2GHz
L1 I&D Caches	64KB each, 2-way, 32B lines
L2 cache	Unified, 1MB 4-way, 64B line
L1 latency	2 cycles
L2 latency	10 cycles
Data, Instr. & Node TLBs	Each 4-way, 32 sets
Page size	4 KB
Memory latency	100 cycles first chunk
Memory bus	500MHz, 8-B wide (4 GB/s)
Fetch / Decode width	4 / 4
Issue / Commit width	4 / 4
Load store queue size	64
Register update unit size	128
Hash engine latency	80 cycles
Hash engine throughput	4 GB/s
Hash algorithm	SHA-1 (truncated to 16B)
Tree arity / node size	4 / 16B
Hard drive throughput	100 MB/s

data defined by their image (both around 190 MB) and thus have the largest overheads (approximately 64 MB). The overheads caused by the TMU-managed trees are on average three orders of magnitude lower than the ~4 GB overhead caused by VAS trees. Figure 9 shows that the time taken for initialization closely follows the memory overheads since every memory block covered by the tree must be processed during initialization. Once again, the TMU dramatically reduces the overheads, by three orders of magnitude on average.

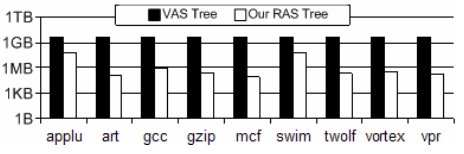


Fig. 8. Initialization-Time Memory Overheads. Average reduction with RAS: 3 orders of magnitude.

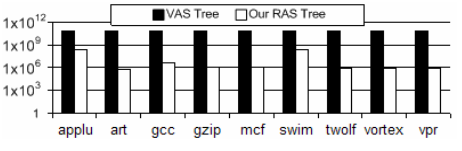


Fig. 9. Initialization latencies. Average reduction with RAS: 3 orders of magnitude.

Figure 10 shows the TMU and its RAS can accommodate 64-bit address spaces with insignificant increase in memory overhead, while the VAS tree overhead is unmanageable. It compares the memory overheads of the two schemes with `gcc` as an example, for different memory spans. While the overhead of dynamic trees managed by the TMU is constant, that of VAS trees increases with the memory span, going to several exabytes (1 exabyte = 2^{60} bytes) for a 64-bit memory span! Such an overhead clearly prevents the scalability of VAS trees to larger virtual address spaces—unlike our RAS tree.

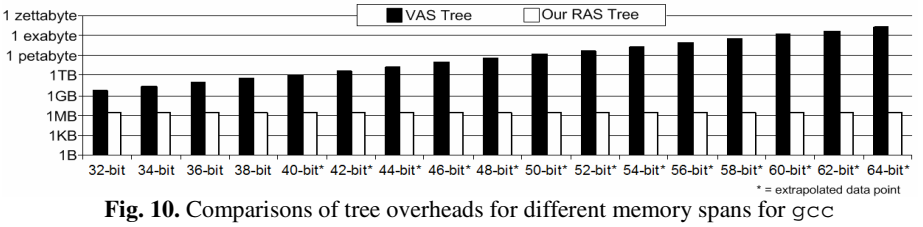


Fig. 10. Comparisons of tree overheads for different memory spans for `gcc`

6.2 Runtime

The size of VAS trees remains the same from initialization to completion of the application's execution. The size of RAS trees however increases with the application's dynamic memory needs. Figure 11 compares the memory overhead required for the peak size RAS tree versus that of a VAS tree. It shows that even at its peak size, the TMU-managed tree is still, on average, three orders of magnitude smaller than the equivalent VAS tree.

We also ran numerous performance simulations to evaluate the impact of our method on the runtime execution performance, measured in Instructions executed Per Cycle (IPC). The IPC count with our TMU implementing RAS memory authentication is normalized to the IPC of an application without any integrity tree protection.

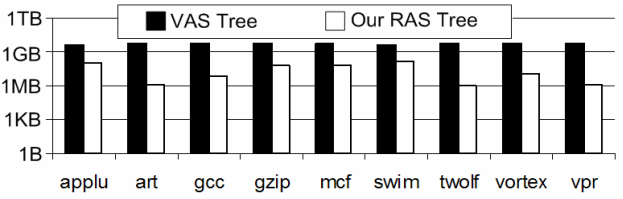


Fig. 11. Runtime Memory Overheads. Average reduction with RAS: 3 orders of magnitude.

No performance degradation is indicated by an IPC of 1. The results have been obtained by simulating 500 million instructions on Simple-Scalar, after having skipped 1.5 billion instruc-tions, to obtain steady-state performance. We studied how varying hardware implementation, parameters, like cache size, page size, memory bandwidth, and disk bandwidth affect the relative performance of VAS and RAS trees.

On the average, a TMU-managed tree degrades the IPC count by 5.15% with respect to no integrity tree and by 2.50% with respect to a VAS tree, an acceptable performance hit in most cases. For the latter, although the height of a RAS-tree is always smaller than that of a VAS tree, the TMU seldom outperforms IPC performance for a VAS tree, due to the caching of tree nodes (in both VAS and RAS trees). Because the microprocessor is considered our hardware security perimeter, data in on-chip caches inside the microprocessor, including cached and validated tree nodes, are considered trusted and safe from attacker manipulation. This caching of tree nodes effectively makes lower level nodes into local sub-tree roots that allow for quick verifications in tall trees.

Considering the extra work needed to add pages and grow the RAS tree during runtime execution, our performance evaluations show that the runtime performance overhead is negligible, while achieving huge benefits in memory overhead and initialization overhead.

7 Conclusion

In this paper, we have shown that PAS trees (integrity trees over physical address spaces) are vulnerable to branch splicing attacks. We described in detail the branch splicing attack to which PAS trees are susceptible but VAS trees (integrity trees over virtual address spaces) are immune to. We explained why VAS trees generate overheads (in space and time) so large as to deter their deployment for memory authentication in general-purpose computers. We then introduced the concept of a dynamically expandable integrity tree, spanning a novel Reduced Address Space (RAS). One novel aspect of our RAS tree is that it keeps track of both used and unused pages—we save space by combining unused pages as Unused Page Ranges (UPRs), and providing full integrity tree coverage to used pages. Our RAS tree is efficiently managed by our proposed Tree Management Unit (TMU). At runtime, our TMU maps into the Reduced Address Space (RAS) only those pages needed by the application and builds an integrity tree over this dynamically expanding RAS. Compared to existing VAS trees, our solution decreases the initialization latency and the memory overheads by two to three orders of magnitude for 32-bit virtual address spaces (VAS), with only a 2.5% hit on the runtime IPC count, in spite of the extra work in growing the RAS tree. As opposed to VAS trees, our RAS solution can easily scale up to provide memory authentication for large 64-bit virtual address spaces.

References

1. Blum, M., Evans, W., Gemmell, P., Kannan, S., Noar, M.: Checking the correctness of memories. Algorithmica 12(2/3), 225–244 (1994)
2. Burger, D., Austin, T.M.: The SimpleScalar Tool Set, Version 2.0., Technical report, University of Wisconsin-Madison Computer Science Department (1997)
3. Elbaz, R., Champagne, D., Lee, R.B., Torres, L., Sassatelli, G., Guillemin, P.: TEC-Tree: A Low Cost and Parallelizable Tree for Efficient Defense against Memory Replay Attacks. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 289–302. Springer, Heidelberg (2007)
4. Gassend, B., Clarke, D., van Dijk, M., Devadas, S., Suh, E.: Caches and Merkle Trees for Efficient Memory Authentication. High Performance Computer Architecture (HPCA-9) (February 2003)
5. González-Barahona, J.M., Ortuño Pérez, M.A., Quirós, P.H., González, J.C., Olivera, V.M.: Counting potatoes: the size of Debian 2.2 (2002), http://people.debian.org/~jgb/debian-counting/counting-potatoes/
6. Hall, W.E., Jutla, C.S.: Parallelizable Authentication Trees. Selected Areas in Cryptography, pp. 95–109 (2005)
7. Hatton, L.: Estimating source lines of code from object code: Windows and Embedded Control Systems (2005), http://www.leshatton.org/LOC2005.html
8. Henning, J.L.: SPEC CPU2000: Measuring CPU performance in the new millenniumm. IEEE Computer (July 2000)
9. I.B.M.: IBM Extends Enhanced Data Security to Consumer Electronics Products. IBM (April 2006), http://www-03.ibm.com/press/us/en/pressrelease/19527.wss
10. Intel, Intel Trusted Execution Technology: Preliminary Architecture Specification (November 2006), http://www.intel.com
11. Kannan, K., Telang, R.: Economic analysis of market for software vulnerabilities. In: Workshop on Economics and Information Security, Minneapolis, MN, USA (May 2004)
12. Lee, R.B., Kwan, P.C.S., McGregor, J.P., Dwoskin, J., Wang, Z.: In: Architecture for Protecting Critical Secrets in Microprocessors, Int'l Symposium on Computer Architecture (ISCA-1932), pp. 2–13 (June 2005)
13. Lie, D., Thekkath, C., Mitchell, M., Lincoln, P., Boneh, D., Mitchell, J., Horowitz, M.: Architectural Support for Copy and Tamper Resistant Software. In: Int'l Conf. on Architectural Support for Programming Languages and OS (ASPLOS-IX), pp. 168–177 (2000)
14. Merkle, R.C.: Protocols for Public Key Cryptosystems. In: IEEE Symposium on Security and Privacy, pp. 122–134 (1980)
15. Rogers, B., Rogers, B., Chhabra, S., Solihin, Y., Prvulovic, M.: Using Address Independent Seed Encryption and Bonsai Merkle Trees to Make Secure Processors OS- and Performance-Friendly. In: Proc. of the 40th Annual IEEE/ACM International Symposium on Microarchitecture, pp. 183–196 (2007)
16. Shi, W., Lu, C., Lee, H.S.: Memory-centric Security Architecture. In: 2005 International Conference on High Performance Embedded Architectures and Compilers (2005)
17. Suh, G.E., Clarke, D., Gassend, B., van Dijk, M., Devadas, S.: AEGIS: Architecture for Tamper-Evident and Tamper-Resistant Processing. In: Proc. of the 17th Int'l Conf. on Supercomputing (ICS) (2003)
18. Yan, C., Rogers, B., Englender, D., Solihin, Y., Prvulovic, M.: Improving Cost, Performance, and Security of Memory Encryption and Authentication. In: Int'l Symposium on Computer Architecture (ISCA-1933), pp. 179–190 (June 2006)

Appendix A: Memory Integrity Trees

Existing techniques preventing the active attacks presented in our threat model are based on tree structures [1, 3, 6, 14]. The common philosophy behind these methods is to split the memory space to protect into N equal size blocks which are the leaf nodes of a balanced A-ary integrity tree. The remaining tree levels are created by recursively applying a function f—which we call the authentication primitive—over A-sized groups of memory blocks, until the procedure yields a single node called the root of the tree. The root reflects the current state of the memory space; making the root tamper-resistant thus ensures tampering with the memory space can be detected. The tree in Figure 1 is a 2-ary (binary) integrity tree.

Tree Authentication Procedure. For each memory block B, there exists a branch, starting at B and ending at the root, composed of the tree nodes obtained by recursive applications of f on B. When B is fetched from untrusted memory, its integrity is verified by recomputing the tree root using the fetched B and the nodes—obtained from external memory—along the branch from B to the root. We know B has not been tampered with when the recomputed root is identical to the on-chip root.

Tree Update Procedure. When a legitimate modification is carried out over a memory block B, the corresponding branch, including the tree root, is updated to reflect the new value of B. This is done by first authenticating the branch B belongs to, then computing on-chip the new values for the branch nodes, and storing the updated branch and storing the tree root on-chip.

Merkle Tree. In a Merkle Tree [1, 14], f is a cryptographic hash function; the nodes of the tree are thus simple hash values. The generic verification and update procedures described above are applied in a straightforward manner. The root of this tree reflects the current state of the memory space since the collision resistance property of the cryptographic hash function ensures that in practice, the root hashes for any two memory spaces differing by even one bit will not be the same. The integrity trees in [3, 6], use different cryptographic primitives for f, but apply the same principles.

Cached Integrity Trees. [4] proposes to cache tree nodes in the on-chip L2 cache. Upon fetching a tree node from main memory, it is checked, stored in L2 and trusted for as long as it is cached on-chip. As a result, during tree authentication and tree update procedures, the first node encountered in cache serves as a local root and terminates the procedures.

An Efficient PIR Construction Using Trusted Hardware

Yanjiang Yang[1,2], Xuhua Ding[1], Robert H. Deng[1], and Feng Bao[2]

[1] School of Information Systems, Singapore Management University, Singapore 178902
[2] Institute for Infocomm Research, Singapore 119613
{yyang,baofeng}@i2r.a-star.edu.sg,
{xhding,robertdeng}@smu.edu.sg

Abstract. For a private information retrieval (PIR) scheme to be deployed in practice, low communication complexity and low computation complexity are two fundamental requirements it must meet. Most existing PIR schemes only focus on the communication complexity. The reduction on the computational complexity did not receive the due treatment mainly because of its $O(n)$ lower bound. By using the trusted hardware based model, we design a novel scheme which breaks this barrier. With constant storage, the computation complexity of our scheme, including offline computation, is linear to the number of queries and is bounded by $O(\sqrt{n})$ after optimization.

1 Introduction

Private Information Retrieval (PIR) was first formulated by Chor et al. in [5]. It offers a strong privacy assurance since it disallows any leakage of user query information. Although PIR should be the ideal privacy guardian of commercial database users, this did not happen. The reason is its prohibitively high cost, as pointed out by Sion and Carbunar [12]. Their analysis shows that a carefully designed PIR scheme with sophisticated cryptographic techniques costs even more time delay than the most trivial solution of transferring the entire database. The culprit for this unexpected effect is the expensive computation cost, which dominates the overall time delay. In the standard PIR model, the lower computation bound is obviously $O(n)$ where n is the database size. A new model based on trusted hardware was introduced in [7,8], which has a logarithm communication complexity and constant online computational complexity. Nonetheless, those schemes are not practical either, since they have to periodically shuffle the *entire* database. Considering the scale of modern databases, a full database shuffle disrupts the database service.

The objective of this paper is to narrow the gap between the ideality and the practicability of PIR. We construct a practical PIR scheme using the same trusted hardware model as in [7,8,14]. With a constant storage cost of the trusted hardware, our construction requires $O(\log n)$ communication cost and $O(\sqrt{n})$ computation cost per query including constant online computation and amortized offline computation.

T.-C. Wu et al. (Eds.): ISC 2008, LNCS 5222, pp. 64–79, 2008.

Related Work. Many PIR constructions were proposed to reduce the communication complexity, including [4,11,9,10,2]. As shown in [13,7,8,14], the communication complexity can be reduced to $O(\log n)$ by using a trusted hardware embedded in the database server. In this model, a trusted hardware is able to perform encryptions/decryptions and has a secret cache of reasonable size for storing retrieved data items. Further advantage of this type of schemes is the $O(1)$ online computation cost for each query. However, all of them require a database re-encryption and re-shuffle whenever the cache is full. Since the available space in the cache decreases linearly with the number of queries, a full database shuffle is performed frequently which requires $O(n)$ operations.

The previous work focusing on computation cost reduction is [3], where Beimel et al. proposed a new model called PIR with Preprocessing. This model uses k servers each storing a copy of the database. Before a PIR execution, each server computes and stores polynomially-many bits regarding the database. This approach reduces both the communication and computation cost to $O(n^{1/k+\epsilon})$ for any $\epsilon > 0$. However, it requires a storage of a polynomial of n bits, which is infeasible in practice. A recent scheme [15] improves the communication and computation complexity to $O(\log^2 n)$ with a cache storing $O(\sqrt{n})$ records.

We notice that the trusted hardware based PIR model is similar to the model in ORAM [6]. But we stress that the "square root" complexity in [6] and our result are in different context. The square root solution of ORAM requires a sheltered storage storing $\sqrt{n}$ words, which is equivalent to using a cache storing $\sqrt{n}$ items in the PIR model. Our scheme in this work, however, only uses a *constant* size cache.

Roadmap. We define the system model and the security notion of our scheme in Section 2. A basic construction is presented in Section 3 as a steppingstone to the full-fledged scheme in Section 4. Performance of our scheme is discussed in Section 5, and Section 6 concludes the paper.

2 System Model and Definition

System Model. Our scheme follows the trusted hardware model used in [8,7,14]. The architecture consists of a group of users, a database D modeled as an array of n data items of equal length denoted by $d_1, d_2, \cdots d_n$, respectively, and a database host H where D is stored. A trusted component denoted by T is embedded in H. To retrieve d_i, a user sends to T a query specifying the index i. T then interacts with H which operates over the encrypted database, and at the end of the execution, T returns d_i to the user. We assume that the communication channel between users and T is confidential.

T is a tamper-resistant hardware with a cache for storing up to k data items, $k \ll n$. No other entity (except T itself) is able to tamper T's protocol executions or access its private space including the cache. T is capable of performing certain

cryptographic operations, such as symmetric key encryptions/decryptions and pseudo-random number generation. In practice T can be implemented by using a specialized trusted hardware such as IBM PCIXCC [1].

Security Definition. The adversary in our model is the database host H, which has polynomial-bounded computational power, and attempts to derive information from the PIR protocol execution. The adversary is able to not only observe accesses to its space, including all read/write operations on the database, but also query the database like a legitimate user.

We use **access pattern** to describe the information observed by the adversary within a time period of protocol execution. When T accesses H's space including the memory and disks, H observes the data in use and the involved positions. The access pattern of length $m \geq 1$ is defined as a sequence of m elements $\langle \alpha_1, \cdots, \alpha_m \rangle$, where each α_i represents the information observed by H during an access to H's space. We use $\mathbb{A}_D$ to denote the set of all possible access patterns generated by querying the database D.

The security notion in ORAM [6] is used here to measure the information leakage from PIR query executions. A secure PIR scheme should ensure that the adversary does not gain additional information about the queries from the access pattern, except the a-priori information. This notion is similar to *perfect secrecy* defined for ciphers where an adversary obtains no additional information about the plaintext from the ciphertext. More formally, let Q be the random variable representing a user query, whose value is the index of the requested item, denoted by $q \in [1, n]$. $\Pr(Q = q)$, or simply $\Pr(q)$, denotes the probability that a query is on d_q. Then, the notion of privacy is defined as:

Definition 1. *A PIR scheme is secure, iff for every database of $n \in \mathbb{N}$ items, given a user query, for every valid access pattern, the conditional probability for the event that the query is on any index q is the same as its a-priori probability, i.e.* $\forall D = \{d_1, \cdots, d_n\}, \forall q \in [1, n], \forall \mathcal{A} \in \mathbb{A}_D, \Pr(Q = q|\mathcal{A}) = \Pr(Q = q)$. □

3 Basic Construction

3.1 Overview

We briefly recall the idea of the schemes in [8,7,14] which are the predecessor of ours. During the system *initialization*, the database is encrypted and permutated by a trusted authority. All subsequent retrievals are operated upon the encrypted database. The database service is provided in *sessions*. A session starts when T's cache is empty and ends when it is full. During a session, T retrieves the requested item from the database if it is not in the cache; otherwise, a random item is read. At the end of the session, the entire database is shuffled, and then a new session commences. The objective of database shuffles is to re-mix the touched records within the database, so that the database host has no idea whether a record in the newly shuffled database has ever been read.

We observe that shuffling the entire database is not indispensable, as long as user queries generate access patterns of identical distribution. Based on this

observation, we in in this work propose a new PIR scheme with *partial* shuffles, where only those records that have ever been accessed are shuffled. We also design a novel *twin retrieval* method, which forces user queries to generate access patterns of the same distribution. A conceptual view of the protocol execution is as follows. A record is labeled *black* if it has ever been accessed. Otherwise, it is *white*. During the system initialization, T generates a secret key sk for a semantically secure cipher, and a secret random permutation $\sigma : [1, n] \rightarrow [1, n]$. Every item d_i in D and its index i are encrypted under sk and written into the $\sigma(i)$-th position of D_0 as a record. In the rest of the paper, we refer to an entry in the original database D and its location as item and index, and refer to an entry in the encrypted database and its location as record and position. We use d_i to denote the i-th item in D, and a_i to denote the i-th record in the shuffled database. After D_0 is generated, all records in D_0 are initially white.

Our PIR service also proceeds in sessions, and the encrypted database in the s-th session is denoted by D_s. During a session, for each user query T executes a *twin retrieval*: if the requested item d_i is in the cache, T reads one random black record and one random white record from D_s; otherwise, T reads the corresponding record and reads one random record in a different color. After the cache is full, T then generates a new random secret permutation π_{s+1} for all black records and updates D_s into D_{s+1} by shuffling and re-encrypting all black records. Those white records remain intact. After the *partial shuffle*, H only knows that a black record has ever been read, but does not know in which session and how many times it has been accessed.

The key problem in implementing this approach is how T securely decides whether a record is black or white. While the label bits of the black records are set, T can not directly access H to check those bits since the access implicates that those records are sought by T. In the following, we assume that T's cache is big enough to accommodate the positions of all black records, so as to facilitate better understanding the idea of our new PIR approach. We remove this assumption in Section 4 by introducing an improved construction.

3.2 A Basic PIR Scheme

We use an array B to keep the black positions in an ascending order. If a_x is a black record and $B[i] = x$, we say that i is the B-Index of a_x. B is stored in H's space and maintained by H: whenever a record is read, it updates B. We use B_s to denote B's state in the beginning of the s-th session. T copies B into its cache before a session starts. During a session, B is updated, whereas T's copy is not changed. Note that for each record read into the cache, T needs to store the corresponding data item and its index in the cache. We denote the cache content by $\mathcal{C}$ and use $\mathcal{C}.Ind$ to denote the set of all stored indices.

A permutation π_s, $s \geq 1$, specifies the mapping between the sets of black positions in D_s and D_0. It is essentially a permutation of B-indexes of all black records. Let $\mathbb{Z}_{|B|} = \{1, 2, \cdots, |B|\}$. Formally, the permutation $\pi_s : \mathbb{Z}_{|B|} \rightarrow \mathbb{Z}_{|B|}$, is defined as: $\pi_s(i) = j$ *if and only if* $D_s[B_s[j]]$ *and* $D_0[B_s[i]]$ *contain the same item, which is* $D[\sigma^{-1}(B_s[i])]$. Note that σ is a mapping between all entries in

D_0 and D. The relations among these notations are $D \stackrel{\sigma}{\Longrightarrow} D_0 \stackrel{\pi_s}{\longrightarrow} D_s$. With B_s, π_s and σ, we are able to locate a record in D_s for a given item index. The PIR protocol proceeds in sessions shown below.

Session 0. T executes k queries using the retrieval algorithm in [14]. Specifically, for a query on the i-th item of D, $i \in [1, n]$, if the requested one is not in T's cache $\mathcal{C}$, T reads the $\sigma(i)$-th record from D_0 into $\mathcal{C}$. Otherwise, T retrieves a random record. At the end of the session, T generates a new random secret permutation $\pi_1 : [1, k] \to [1, k]$. It shuffles the k black records according to π_1 while leaving the white records intact. Since all records to be shuffled are in $\mathcal{C}$, T simply re-encrypts and writes them out sequentially to generate D_1, and clears the cache.

Session $s \geq 1$. When session s starts, $\mathcal{C}$ is empty. T processes $k/2$ queries in the session. For a user query, T executes Algorithm 1 shown below. At the end of the session, T executes Algorithm 2 to shuffle all black records.

Algorithm 1. Basic Twin Retrieval Algorithm in Session $s \geq 1$. **Input:** a query on i, $i \in [1, n]$, $B_s[1, (s+1)k/2]$. **Output:** the item $d_i \in D$.

1: Through the secure channel, T accepts a query from the user requesting the i-th item in D.
2: **if** $i \notin \mathcal{C}.Ind$ **then**
3: $j = \sigma(i)$.
4: binary-search j in B_s; /*we do not elaborate the binary-search algorithm since it is a standard one*/
5: **if** exists u, s.t. $B_s[u] = j$ **then**
6: d_i is black; Read $D_s[B_s[\pi_s(u)]]$ as d_i and read a random white record;
7: **else**
8: d_i is white; read a random black record and read $D_s[j]$ as d_i;
9: **end if**
10: **else**
11: read a random black record and a white record from D_s into $\mathcal{C}$.
12: **end if**
13: return d_i to the user.

We now explain the retrieval algorithm (Algorithm 1) and the shuffle algorithm (Algorithm 2). In Algorithm 1, T searches B_s to determine the color of the requested record. For a white record, T directly uses its image under σ to read the data, since it has never been shuffled. For a black records, T computes its B-index under π_s and then looks up B_s to locate its position in D_s. Since B_s is inside T's cache, all accesses are not visible to the server. For a query execution, H only observes one read to a black record and one read to a white record. After $k/2$ queries, the cache is full, where half are black and half are white. B maintained by H now has $(2 + s)k/2$ entries.

The partial shuffle is to mix the black records including those newly retrieved during the session, so that they are randomly relocated in D_{s+1}. The basic idea of the algorithm is the following: T updates the black positions in D_s sequentially. For each black position, T figures out the location of its preimage under π_{s+1}.

Algorithm 2. Basic Partial Shuffle Algorithm executed by T at the end of s-th session, $s \geq 1$. **Input:** B with $(2+s)k/2$ black records, cache $\mathcal{C}$ with $k/2$ black and $k/2$ white records; **Output:** D_{s+1}.

1: scan B. For each item in the cache, calculate its index in B.
2: secretly generate a random permutation $\pi_{s+1} : \mathbb{Z}_{|B|} \rightarrow \mathbb{Z}_{|B|}$.
3: **for** $(i = i' = 1; i \leq sk/2; i++)$ **do**
4: $\quad j = \pi_{s+1}^{-1}(i')$;
5: $\quad$ **while** $\sigma^{-1}(B_s[j]) \in \mathcal{C}.Ind$ and $i' \leq sk/2$ **do**
6: $\quad\quad i' = i' + 1$; $j = \pi_{s+1}^{-1}(i')$; /**find one not from* $\mathcal{C}$**/*
7: $\quad$ **end while**
8: $\quad$ count δ as the number of white indexes in $\mathcal{C}$ which are smaller than j,
9: $\quad$ compute $v = \pi_s(j - \delta)$; read $D_s[B_s[v]]$.
10: $\quad$ **if** $i \neq i'$ **then**
11: $\quad\quad$ Re-encrypt $D_s[B_s[v]]$ into $D_{s+1}[B[i]]$;
12: $\quad$ **else**
13: $\quad\quad$ Insert $D_s[B_s[v]]$ to cache. Retrieve the corresponding item from $\mathcal{C}$ and re-encrypt it to $D_{s+1}[B[i]]$.
14: $\quad$ **end if**
15: $\quad i' = i' + 1$;
16: **end for**
17: write the remaining k records in $\mathcal{C}$ to D_{s+1} accordingly, securely eliminate π_{s-1}.
18: copy B into the cache as B_{s+1}. End the session.

If the preimage is in $\mathcal{C}$, T finds the next position whose preimage is not in $\mathcal{C}$ (as shown in Step 5, 6, 7). The computation of the preimage location involves the composition of π_{s+1}^{-1} and π_s. Since π_{s+1}^{-1}'s range is larger than π_s's domain, a translation from an index in B_{s+1} to B_s is needed (Step 8). As B_{s+1} is actually a combination of sorted B_s and the white positions (positions of newly retrieved white records) in sorted $\mathcal{C}$, we are ensured that $B_{s+1}[i] = B_s[i - \delta]$ (Step 8), where δ is the number of white indices in $\mathcal{C}$ smaller than i. The average cost of finding δ is $\mathrm{O}(\log k)$ (The cost can be reduced to $\mathrm{O}(1)$ by keeping two copies of B in the cache and using pointers to link them.). Among the variables used in Algorithm 2, $B[i]$ points to the black position in D_{s+1} for writing whereas $B_s[i']$ points to the black position in D_s for reading. None of them decreases. Therefore, the overall complexity is $\mathrm{O}(sk \log k)$.

3.3 Security Analysis

Due to the length limit, we only formalize the security of our scheme by presenting the following lemmas, whose proofs are available in Appendix. Lemma 1 shows that the basic partial shuffle (Algorithm 2) is uniform in the sense that after the partial shuffle at the end of Session s, the previous black records in D_s and the white records retrieved during the session are randomly re-located to D_{s+1}. Thus, all black records appear indistinguishable to H. Then, Lemma 2 claims that at any time, the access patterns for any two queries of the basic twin retrieval algorithm (Algorithm 1) have the same distribution. Finally, by

the results of Lemma 1 and Lemma 2, we prove in Theorem 1 that the basic PIR scheme is secure, satisfying Definition 1.

Lemma 1 (Uniform Shuffle). *The basic partial shuffle algorithm performs a uniform shuffle on all black records. Namely,* $\forall s > 0$, $\forall \jmath, i \in B_s$,

$$\Pr(D_s[\jmath] \simeq D_0[i] \,|\, \mathcal{A}_0, \mathcal{R}_0, \cdots, \mathcal{A}_{s-1}, \mathcal{R}_{s-1}) = 1/|B_s|,$$

where $\mathcal{A}_l$ *and* $\mathcal{R}_l$, $l \in [0, s-1]$ *are the access pattern and the reshuffle pattern for the* l*-th session, respectively.* $D_s[\jmath] \simeq D_0[i]$ *means* $D_s[\jmath]$ *and* $D_0[i]$ *have the same plaintext.*

Lemma 2 (Uniform Access). *Let* Q *be the random variable for the requested item's index in* D*. Let* (X, Y) *be the two-dimensional random variable for the positions of the black record and the white record accessed in the twin retrieval algorithm corresponding to* Q*.* $\forall q_1, q_2 \in [1, n]$*, suppose* $\mathcal{A}$ *is the access pattern when* $Q = q_1$ *or* $Q = q_2$ *is executed, then* $\Pr((X = x, Y = y) \mid \mathcal{A}, Q = q_1) = \Pr((X = x, Y = y) \mid \mathcal{A}, Q = q_2)$.

Theorem 1 (Security of PIR). *Let* $\mathcal{A}_K$ *be the access pattern of* K *database accesses. For query* Q, $\forall q \in [1, n]$, $\forall K \in \mathbb{N}$, $\forall \mathcal{A}_K$, $\Pr(Q = q \mid \mathcal{A}_K) = \Pr(Q = q)$.

4 A Construction without Storage Assumption

In this section, we propose an improved scheme without assuming T's capability in storing B. As we mentioned earlier, the exposure of accesses to B leads to security breaches, since it indicates that the accessed ones are entries pertaining to the query in execution. Informally, the access to B requires a PIR-like solution. A trivial solution is to treat B as a database and to run a PIR query on it. Surely, the cost of this approach seriously counteracts our effort to improve the computational efficiency. We design a much more efficient solution due to the fact that T has the prior knowledge of those accesses.

4.1 Auxiliary Data Structures

Management of Black Positions. Recall that D_s is a result of a partial shuffle under the permutation $\pi_s : \mathbb{Z}_{|B|} \to \mathbb{Z}_{|B|}$. We use $|B|$ pairs of tuples $\langle x, y \rangle$ to represent this mapping, where $x \in \mathbb{Z}_n$ is a position in D_0 and $y \in \mathbb{Z}_n$ is the corresponding position under π_s in D_s. T selects a *deterministic* symmetric key encryption scheme $e(\cdot)$ and a secret key u. Let $f_u(x) = \mathcal{H}(e_u(x))$, where $\mathcal{H}$ is a hash function. These $|B|$ half-encrypted pairs are stored in an sorted array $L = [(f_u(x_1), y_1), (f_u(x_2), y_2) \cdots, (f_u(x_{|B|}), y_{|B|})]$, where $y_1 < \cdots < y_{|B|}$. Note that the sequence of y-values in L is exactly array B, which explains why we leave y-values in plaintext. However, B is updated by H due to query executions whereas L is not. We also build a *complete binary search tree* Γ where the tuples in L are the leaves in the following manner: from left to right, the leaves are

sorted in an ascending order of $f_u(x)$ values. All the $|B|-1$ inner nodes are integers randomly assigned by T according to their left and right children.

T makes use of L and Γ to decide whether an item is a white or black record, and to read a specific or random black record.

- To read an item with index x: If $f_u(\sigma(x))$ is smaller than the leftmost leaf or larger than the rightmost leaf of Γ, T immediately knows that $\sigma(x)$ is a white position. Otherwise, it runs a binary search for $f_u(\sigma(x))$ in Γ. Suppose that the search ends at a leaf node $\langle f_u(x'), y\rangle$. If $f_u(x) = f_u(x')$, y is the position of the requested item; otherwise, y is taken as a randomly selected black position.
- Random search: Starting from the root, T tosses a coin at each level to select either the left child or the right child as the next node to read. In the end, it returns a leaf.

L and Γ are (re)constructed at the end of each session. L is initialized when T executes the partial shuffle under π_s whose algorithm is explained Section 4.2. During a shuffle, T sequentially writes to those positions stored in B, which is exactly y-values in L. Therefore, for each data item d_i relocated to the black position stored at $B[r]$, T sets $L[r] = \langle f_u(\sigma(i)), B[r]\rangle$, where u is a new encryption key. Once L is established, construction of Γ is straightforward.

Management of White Positions. We need to manage white records as well. The $|B|$ black records virtually divide the database into white segments, i.e. blocks of adjacent white records. We use an array $W[\,]$ in H's space to sequentially store these white segments, such that $W[i] = \langle l, m, M\rangle$ indicating that the i-th white segment in the database starts from the record a_l and contains m white records. We set $W[i].M = \sum_{j=1}^{i-1} W[j].m + 1$ such that a_l is the $W[i].M$-th white record in the database. Different from L and Γ, W is managed by H. T makes use of W to read white records in the following manner.

- To read the white record with index x: T runs a binary search on W for the address $\sigma(x)$, such that it stops at $W[i]$, such that $W[i].l \le \sigma(x) < W[i+1].l$. Then, it reads the $\sigma(x)$-th records from D_s.
- Random search: T generates $r \in_R [1, n-|B|]$. Then it runs a binary search on W for the r-th white record in D_s, such that it stops at $W[i]$, such that $W[i].M \le r < W[i+1].M$. Finally, it returns $y = W[i].l + r - W[i].M$.

For both cases, H only observes two search paths, which H cannot differentiate the two types of retrievals.

We need to store more information in $\mathcal{C}$ as well. Suppose that T retrieves a record a_j into $\mathcal{C}$. A new entry is created as a tuple (*BIndex, Color, Ind,Data*) where Ind and *Data* are the corresponding item's index and value, respectively; *Color* is set to 'B' if a_j was black before retrieval; otherwise Color is set to 'W'; $BIndex$ is set to a_j's B-Index with respect to D_0. We use $\mathcal{C}[i]$ to denote the i-th entry of the cache, $\mathcal{C}.Ind$ to denote the set of all entries' *Ind* values, $\mathcal{C}.BIndex$ to denote the set of all entries' *BIndex* values. Note that B_s is no longer used, as B is not stored in $\mathcal{C}$.

4.2 The Improved Scheme

We are now ready to present the scheme. It proceeds in a similar way as the basic scheme in Section 3.2. The difference is that since Session 1, T executes Algorithm 3 for a query execution and Algorithm 4 for the partial shuffle. Algorithm 3 shows how to process a query during the s-th session, $s \geq 1$. Note that $|B| = (s+1)k/2$ when session s starts.

Algorithm 3. Improved Twin Retrieval Algorithm in Session $s \geq 1$, executed by T. **Input:** a user query on $i \in [1, n]$. **Output:** the data item $d_i \in D$.

1: Through the secure channel, T accepts a query from the user requesting for the i-th item in D.
2: $min = B[1]$; $max = B[(s+1)k/2]$; /*(min, max) *is the range of black positions.**/
3: $i' = \sigma(i)$;
4: **if** $i \in \mathcal{C}.Ind$ **then**
5: randomly search Γ which returns $\langle f_u(x), y\rangle$. Then $j_b \leftarrow y$.
6: randomly search W which returns j_w.
7: **else**
8: **if** $i' < min$ or $i' > max$ **then**
9: randomly search Γ which returns $\langle f_u(x), y\rangle$. Then $j_b \leftarrow y$.
10: **else**
11: search Γ for $f_u(i')$ which returns $\langle f_u(x, y)\rangle$. Then $j_b \leftarrow y$.
12: **end if**
13: **if** $f_u(x) = f_u(i')$ **then**
14: randomly search W which returns j_w.
15: **else**
16: search W for i', which returns $\langle l, m, M\rangle$. Then $j_w \leftarrow i'$.
17: **end if** /*Note that $f_u()$ *is deterministic.**/
18: **end if**
19: read the j_b-th and the j_w records from D_s, and creates two new entries for them $\mathcal{C}$ accordingly; return d_i to the user.

Access Pattern of Retrievals. We use $\mathcal{A}_s$ to denote the access pattern produced by Algorithm 3. There are three types of accesses to H's space. The first type is the database accesses. For simplicity, we use the accessed black and white positions, denoted by (α_i, α'_i), as the access pattern in the i-th query execution. The second type is the accesses to W during the searches. The output of a binary search on W determines the involved search path. Furthermore, the output of a search can be derived by observing the subsequent database access. Therefore, the second type of accesses does not reveal extra information. We do not include it in $\mathcal{A}_s$. The third type is the retrieval of elements in Γ. Similar to the previous reasoning, the access to Γ does not divulge extra information and is not included either. According to Algorithm 3, totally $k/2$ queries are executed in a session. Thus, the access pattern produced during the s-th session is $\mathcal{A}_s = \langle \alpha_1, \alpha'_1, \cdots, \alpha_{k/2}, \alpha'_{k/2}\rangle$. The access pattern in Session 0 is an exception, since one record is retrieved per query: $\mathcal{A}_0 = \langle \alpha_1, \cdots, \alpha_k\rangle$.

Algorithm 4 shows how to perform a partial shuffle at the end of the s-th session. Note B has expanded from $(s+1)k/2$ elements to $(s+2)k/2$ elements due to the $k/2$ retrievals of white records in this session. The partial shuffle process requires $(s+2)k/2$ database writes and $sk/2$ database reads. We remark that the computation cost for constructing Γ' is not expensive for the following two reasons. First, those operations are memory based integer comparisons, which are much cheaper than database accesses. Second, the sorting process can be done by H.

Algorithm 4. Improved Partial Shuffle Algorithm executed by T at the end of s-th session, $s \geq 1$. **Input:** cache $\mathcal{C}$ with $k/2$ black and $k/2$ newly retrieved white records; **Output:** D_{s+1}, Γ' and L'.

1: scan B and assign the $BIndex$ field for each entry in $\mathcal{C}$. Specifically, for every $1 \leq b \leq |B|$, if $\exists a \in [1,k]$, s.t. $\sigma(\mathcal{C}[a].Ind) = B[b]$, then set $\mathcal{C}[a].BIndex = b$.
2: generate a secret random permutation $\pi_{s+1} : \mathbb{Z}_{|B|} \rightarrow \mathbb{Z}_{|B|}$, and a new encryption key u'.
3: **for** $(i = i' = 1; i \leq sk/2; i++)$ **do**
4: $\quad j = \pi_{s+1}^{-1}(i')$;
5: $\quad$ **while** $\sigma^{-1}(B[j]) \in \mathcal{C}.Ind$ and $i' \leq sk/2$ **do**
6: $\quad\quad i' = i' + 1$; $j = \pi_{s+1}^{-1}(i')$; /*find one not from $\mathcal{C}$*/
7: $\quad$ **end while**
8: $\quad$ count δ as the number of white indexes in $\mathcal{C}$ which are smaller than j,
9: $\quad$ compute $v = L[\pi_s(j-\delta)].y$; Read $D_s[v]$. Suppose that $D_s[v] = E_{sk}(x, d_x)$.
10: $\quad$ **if** $i' = i$ **then**
11: $\quad\quad$ Re-encrypt $D_s[v]$ into the $D_{s+1}[B[i]]$;
12: $\quad\quad L'[i] \leftarrow \langle f_{u'}(\sigma(x)), B[i] \rangle$;
13: $\quad$ **else**
14: $\quad\quad$ insert a 4-tuple $\langle 0, \text{'B'}, x, d_x \rangle$ into $\mathcal{C}$.
15: $\quad\quad$ find $l \in [1,k]$ satisfying $\mathcal{C}[l].BIndex = \pi_{s+1}^{-1}(i)$. Insert $E_{sk}(\mathcal{C}[l].Ind, \mathcal{C}[l].Data)$ to $D_{s+1}[B[i]]$.
16: $\quad\quad L'[i] \leftarrow \langle f_{u'}(\sigma(\mathcal{C}[l].Ind)), B[i] \rangle$.
17: $\quad$ **end if**
18: $\quad$ i'=i'+1
19: **end for**
20: write the remaining k records in the cache to D_{s+1} and assign L' accordingly.
21: construct Γ' based on L' and discard π_s, L, Γ.

Access Pattern of Shuffles. We use $\mathcal{R}_s$ to denote the access pattern produced by Algorithm 4 at the end of the s-th session. There are three types of accesses. The first type is the accesses to B. However, since all accesses to B are in a predetermined order, they do not leak any information (they can be generated correctly by H without observing the execution). We exclude them from $\mathcal{R}_s$.

The second type is the read and write accesses to the database. According to our algorithm, a read access is always followed by a write access. The sequence of the writes are known to H before the shuffle, since it follows the sequence of positions in B. Furthermore, the contents of the writes are new ciphertext under a semantic secure encryption. Therefore, the access pattern of writes does

not expose information to H. Considering the read pattern only, we use α_i, the position of the i-th read access, to represent the access pattern.

The third type of accesses is the read operations on L and the write operations on L'. Every write to L' is always preceded by a read access to the database. Moreover, the sequence of writings to L' and the contents of L', except those encryptions, can be determined by H without observing the execution. Therefore, the write accesses on L' do not leak information. Every read operation on L exposes the touched index of L and a black position y. However, the exposed black position y can also be determined by observing the subsequent database read. Since L is known to H, knowing the black position y naturally implies the knowledge of its position in L. Thus, it suffices to represent the access pattern only using the database accesses, i.e. $\mathcal{R}_s = \langle \alpha_1, \cdots, \alpha_{sk/2} \rangle$, where $\alpha_i \in [1, n], 1 \leq i \leq sk/2$.

The shuffle at the end of Session 0 is a special case, where all records to be shuffled are in T's cache. T simply writes them out to the corresponding positions following the permutation π_1, in which case, $\mathcal{R}_0 = \emptyset$.

4.3 Security Analysis

Security analysis of the improved scheme is based on that of the basic scheme. By Lemma 3 and Lemma 4, we show that the improved partial shuffle algorithm (Algorithm 4) and the improved twin retrieval algorithm (Algorithm 3) also perform a uniform shuffle and a uniform access, respectively. This in turn implies that Theorem 1 also holds for the improved scheme.

Lemma 3. *Lemma 1 also holds for the improved partial shuffle algorithm (Algorithm 4).*

Proof (sketch)**:** We compare the access patterns of the improved scheme with those of the basic scheme. The analysis in Section 4.1 has shown that the accesses to Γ and L do not leak extra information. Both shuffle patterns have the same distribution, since they are only determined by the permutations in use. Thus the proof for Lemma 1 is also valid for Algorithm 4. □

Lemma 4. *Lemma 2 also holds for the improved twin retrieval algorithm (Algorithm 3).*

Proof (sketch)**:** The only difference between patterns generated by Algorithm 1 and Algorithm 3 is that the latter uses the search of Γ to generate a random black record. Nonetheless, under the random oracle mode, the function $f_u(\cdot)$ outputs a random number. □

5 Scheme Complexity

The communication cost of our scheme remains the same as other hardware-based PIR schemes [8,7,14]. Namely, it requires $O(\log n)$ communication complexity, as the user inputs a $\log n$-bit long index of the desired data item, and T

returns exactly one item of constant length. The database read/write are counted as a part of the computation cost. Note that $O(\log n)$ is the lower bound of communication complexity for any PIR construction.

When considering the computational complexity, we regard an access to the host H's space and a decryption/encryption as one unit of operation. In the s-th session, a query retrieval using Algorithm 3 requires $O(\log(sk))$ operations due to the task of a binary search in Γ. A partial shuffle at the end of the s-th session requires $O(sk \log k)$ operations. Thus, the overall computation cost in all s-th sessions is $O(s^2 k \log k)$ for totally $(2+s)k/2$ query executions. Consequently, the average cost per query for s sessions is $O(s \log k)$, which is independent of the database size.

When the session number s reaches the order of n, the advantage of our scheme diminishes. A remedy to this problem is to shuffle the entire database at the end of the t-th session. The full shuffle resets the system to its initial state (Session 0). All records are colored white again, as the traces of all previous accesses are removed. Since an early full shuffle might not be able to fully exploit the benefits of partial shuffles, it is necessary to determine an optimal t. Recall that a full shuffle costs $O(n)$ operations. With a full shuffle for every t sessions, the total cost for s sessions becomes $O((t^2 k \log k + n)s/t)$ and the average cost per query is $O(t \log k + n/tk)$ which is minimal when $t \log k = n/tk$. Therefore, the optimal value for t is $\sqrt{\frac{n}{k \log k}}$. The cost per query becomes $O(\sqrt{\frac{n \log k}{k}})$.

A comparison of our scheme against other PIR schemes is given in Table 1. Note that all previous hardware-based schemes [7,8,13,14] claim $O(1)$ computation complexity since they only count the cost of database accesses. In fact, all of them requires $O(\log k)$ operations to determine if an item is in cache. Our scheme also has $O(1)$ database read/write, though we need an additional cost for a binary search in Γ. For those PIR schemes without using caches, the computation cost per query is at least $O(n)$. From the table, it is evident that our scheme substantially outperforms all other PIR schemes in terms of average query cost by paying a slightly higher price of online query processes.

Table 1. Comparison of Computation Performance

Schemes	**Cost of online query process**	**Average overall cost per query**
Our scheme	$O(\log s + \log k)$	$O(s \log k)$
Our scheme with full-shuffles	$O(\log s + \log k)$	$O(\sqrt{\frac{n \log k}{k}})$
Scheme in [14]	$O(1)$	$O(n/k)$
Scheme in [7,8]	$O(1)$	$O(\frac{n \log n}{k})$
Scheme in [13]	$O(1)$	$O(n)$
Other PIR schemes without using caches	$O(n)$	$O(n)$

Notations: n: the database size; s: the number of sessions; k: the size of the cache, $k << n$.

6 Conclusion

All existing PIR schemes have $O(n)$ computational cost for each query. In this paper, we broke this barrier using a novel approach for database retrieval and shuffle. The average cost per query is reduced to $O(s)$ where s is the number of queries, or $O(\sqrt{n})$ in maximum if an optimization is used. We proved the security of our scheme.

Acknowledgement

This research is partly supported by the Office of Research, Singapore Management University.

References

1. Arnold, T., Doorn, L.V.: The ibm pcixcc: A new cryptographic coprocessor for the ibm eserver. Journal of Research and Development 48 (May 2004)
2. Beimel, A., Ishai, Y., Kushilevitz, E., Raymond, J.-F.: Breaking the $o(n^{1/(2k-1)})$ barrier for information-theoretic private information retrieval. In: Proceedings of IEEE FOCS 2002, pp. 261–270 (2002)
3. Beimel, A., Ishai, Y., Malkin, T.: Reducing the servers computation in private information retrieval: PIR with preprocessing. In: Bellare, M. (ed.) CRYPTO 2000. LNCS, vol. 1880, pp. 55–73. Springer, Heidelberg (2000)
4. Chor, B., Gilboa, N.: Computationally private information retrieval. In: Proceedings of the 29th STOC 1997, pp. 304–313 (1997)
5. Chor, B., Kushilevitz, E., Goldreich, O., Sudan, M.: Private information retrieval. In: Proceedings of IEEE FOCS 1995, pp. 41–51 (1995)
6. Goldreich, O., Ostrovsky, R.: Software protection and simulation on oblivious rams. Journal of the ACM 43(3), 431–473 (1996)
7. Iliev, A., Smith, S.: Private information storage with logarithm-space secure hardware. In: Proceedings of International Information Security Workshops, pp. 199–214 (2004)
8. Iliev, A., Smith, S.: Protecting client privacy with trusted computing at the server. IEEE Security & Privacy 3(2), 20–28 (2005)
9. Kushilevitz, E., Ostrovsky, R.: Replication is not needed: single database, computationally private information retrieval. In: Proceeding of the 38th IEEE FOCS 1997, pp. 364–373 (1997)
10. Kushilevitz, E., Ostrovsky, R.: One-way trapdoor permutations are sufficient for non-trivial single-server private information retrieval. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 104–121. Springer, Heidelberg (2000)
11. Ostrovsky, R., Shoup, V.: Private information storage. In: Proceedings of the 29th STOC 1997, pp. 294–303 (1997)
12. Sion, R., Carbunar, B.: On the computational practicality of private information retrieval. In: Proceedings of NDSS 2007 (2007)
13. Smith, S., Safford, D.: Practical server privacy with secure coprocessors. IBM Systems Journal 40(3), 683–695 (2001)
14. Wang, S., Ding, X., Deng, R., Bao, F.: Private information retrieval using trusted hardware. In: Gollmann, D., Meier, J., Sabelfeld, A. (eds.) ESORICS 2006. LNCS, vol. 4189, pp. 49–64. Springer, Heidelberg (2006)
15. Williams, P., Sion, R.: Usable PIR. In: Proceedings of NDSS 2008 (2008)

Appendix

Proof of Lemma 1 (Sketch)**:** This proof is nearly the same as Lemma 1 in [14] with the only difference being what records to be shuffled. An intuitive explanation is that we can treat the black record set pointed by B_s as a database in [14]. There exist two critical points of the proof. 1) For any record in the cache, the probability of a black position in D_s being chosen as the shuffle destination is exactly $\frac{1}{B_s}$. In other words, its image position in B_s is uniformly selected. This is obvious since its is determined by a random π_s. As it is written from the cache, the selection of the position is independent of those access patterns. 2) For any record not in the cache, its preimage position in the previous shuffle was uniformly chosen from black positions in D_{s-1} pointed by B_{s-1}. This is addressed by using induction on s. □

Proof of Lemma 2 (Sketch)**:** Assume that Q is executed at session s. We prove the theorem by examining the cases when $s = 0$ and $s \geq 1$.

I: $s = 0$. The theorem clearly holds as D_0 is a random permutation of D. Therefore, for each instance of Q on D, its image Y on D_0 is uniformly distributed. X is always 0.

II: $s \geq 1$. According to the algorithm, for a query Q, a black record and a white record are read. Define $I = \{i | i \in [1, n], i \in C\}$ containing the indices whose corresponding items are in the cache, and $J_1 = \sigma^{-1}(B_s) \setminus I$ containing the indices of black records, but presently not in the cache, and $J_2 = [1, n] \setminus (I \cup J_1)$, containing the indices whose corresponding items have never been accessed so far. To prove the theorem, it is sufficient to demonstrate that for any $q \in [1, n]$, $\Pr((X = x, Y = y) \mid \mathcal{A}, Q = q)$ remains the same in the following cases covering all possibilities of q.

- Case (1) $q \in J_1$. T reads the corresponding black record and a random white record from D_s. Due to Lemma 1, the corresponding record could be in any position in B_s with the same probability. Therefore $\Pr(X = x \mid \mathcal{A}, q) = \frac{1}{|B_s|}$. Y is a random retrieval, which is independent of $\mathcal{A}$. Therefore, $\Pr((X = x, Y = y) \mid \mathcal{A}, Q = q) = (\frac{1}{|B_s|}, \frac{1}{n-|B_s|})$.
- Case (2) $q \in J_2$. T reads a random black record and the corresponding white record from D_s. The position of the white records is determined by σ. Therefore, $\Pr(Y = y \mid \mathcal{A}, q) = \frac{1}{n-|B_s|}$. X is a random retrieval independent from $\mathcal{A}$. Therefore $\Pr((X = x, Y = y) \mid \mathcal{A}, Q = q) = (\frac{1}{|B_s|}, \frac{1}{n-|B_s|})$.
- Case (3) $q \in I$. Both X and Y are randomly retrieved. So $\Pr((X = x, Y = y) \mid \mathcal{A}, Q = q) = (\frac{1}{|B_s|}, \frac{1}{n-|B_s|})$

This completes the proof. □

Proof of Theorem 1 (Sketch)**:** It is equivalent to prove that $\forall K \in \mathbb{N}$, $\Pr(\mathcal{A}_K \mid Q = q) = \Pr(\mathcal{A}_K)$. Fix a session s, we prove the theorem by using induction on the size of $\mathcal{A}_K$.

I: When $K = 1$, our target is to prove that $\Pr(X = x, Y = y \mid Q = q) = \Pr(X = x, Y = y)$, $\forall x, y \in [n]$.

$\Pr(x, y) = \sum_{i=1}^{n} \Pr(x, y|i)\Pr(i)$. Consider $\Pr(x, y \mid i)$. There are two cases:

- The record corresponding to i is in B_s. Therefore, $\Pr(x) = \frac{1}{|B_s|}$, due to the initial permutation; $\Pr(y) = \frac{1}{n-|B_s|}$ due to random access.
- The record corresponding to i is in $[n] \backslash B_s$. Therefore, $\Pr(x) = \frac{1}{|B_s|}$, due to random access; $\Pr(y) = \frac{1}{n-|B_s|}$ due to the initial permutation.

Thus, in both case $\Pr(x, y \mid i) = (\frac{1}{|\Delta|}, \frac{1}{n-|\Delta|})$ for both cases. Obviously, $\Pr(x, y \mid i) = \Pr(x, y \mid j)$ for all $i, j \in [1, n]$. Consequently, $\Pr(x, y) = \Pr(x, y \mid q) \sum_{i=1}^{n} \Pr(i) = \Pr(x, y \mid q), \forall q \in [1, n]$.

II: Suppose that when $K = k-1$, the equation holds. We then prove that it still holds when $K = k$, i.e. $\Pr(\mathcal{A}_k \mid Q = q) = \Pr(\mathcal{A}_k)$. Without loss of generality, let $\mathcal{A}_k = \mathcal{A}_{k-1} \cup (x, y)$, where $(X = x, Y = y)$ is the k-th database read.

$$
\begin{aligned}
&\Pr(\mathcal{A}_{k-1}, (x, y) \mid q) = \Pr(\mathcal{A}_k) \\
&\frac{\Pr(\mathcal{A}_{k-1}, x, y, q)}{\Pr(q)} = \Pr(\mathcal{A}_{k-1}, (x, y)) \\
&\frac{\Pr(x, y \mid \mathcal{A}_{k-1}, q)\Pr(\mathcal{A}_{k-1}, q)}{\Pr(q)} = \Pr(\mathcal{A}_{k-1}, (x, y)) \\
&\Pr(x, y \mid \mathcal{A}_{k-1}, q) = \frac{\Pr(\mathcal{A}_{k-1}, x, y)}{\Pr(\mathcal{A}_{k-1} \mid q)} \\
&\Pr(x, y \mid \mathcal{A}_{k-1}, q) = \Pr(x, y \mid \mathcal{A}_{k-1}) \\
&(\because \text{induction assumption})
\end{aligned}
$$

Note that there are three exclusive cases for $Q = q$.

1. $Q = q$ occurs after the k-th database access;
2. $Q = q$ is the query for the k-th database access;
3. $Q = q$ occurs prior to the k-th database access.

We proceed to prove that the above equation holds for all three different cases.

Case 1: Obviously, in this scenario, $\mathcal{A}_{k-1}$ and (x, y) are independent of $Q = q$. Therefore, $\Pr(x, y \mid \mathcal{A}_{k-1}, q) = \Pr(x, y \mid \mathcal{A}_{k-1})$.

Case 2: Note that

$$\Pr(x, y \mid \mathcal{A}_{k-1}) = \sum_{q=1}^{n} \Pr(x, y \mid \mathcal{A}_{k-1}, q)\Pr(q \mid \mathcal{A}_{k-1}),$$

where $Q = q$ is the query corresponding (x, y). Due to Lemma 2, $\Pr(x, y \mid \mathcal{A}_{k-1}, q) = \Pr(x, y \mid \mathcal{A}_{k-1}, q')$, $\forall q, q' \in [1, n]$. Therefore,

$$\Pr(x, y \mid \mathcal{A}_{k-1}) = \Pr(x, y \mid \mathcal{A}_{k-1}, q) \sum_{i=1}^{n} \Pr(i \mid \mathcal{A}_{k-1}).$$

According to the induction, $\Pr(i \mid \mathcal{A}_{k-1}) = \Pr(i)$, we have $\Pr(x, y \mid \mathcal{A}_{k-1}) = (x, y \mid \mathcal{A}_{k-1}, q)$.

Case 3: Let Q' be the random variable for the k-th query which generates (x, y). Considering all possible values of Q', denoted by q', we have

$$\Pr(x, y|\mathcal{A}_{k-1}, q) = \sum_{q'=1}^{n} \Pr(x, y \mid \mathcal{A}_{k-1}, q, q')\Pr(q' \mid \mathcal{A}_{k-1}, q)$$

Note that $\Pr(x, y \mid \mathcal{A}_{k-1}, q, q') = \Pr(x, y \mid \mathcal{A}_{k-1}, q')$ since (x, y) is determined by Q' and $\mathcal{A}_{k-1}$ according to our PIR algorithm. Therefore,

$$\Pr(x, y|\mathcal{A}_{k-1}, q) = \sum_{q'=1}^{n} \Pr(x, y \mid \mathcal{A}_{k-1}, q')\Pr(q' \mid \mathcal{A}_{k-1}, q)$$

Since $Q' = q'$ is independent of $\mathcal{A}_{k-1}$ and $Q = q$, thus

$$\begin{aligned}\Pr(x, y|\mathcal{A}_{k-1}, q) &= \sum_{q'=1}^{n} \Pr(x, y \mid \mathcal{A}_{k-1}, q')\Pr(q' \mid \mathcal{A}_{k-1}) \\ &= \Pr(x, y \mid \mathcal{A}_{k-1}).\end{aligned}$$

□

Athos: Efficient Authentication of Outsourced File Systems*

Michael T. Goodrich[1], Charalampos Papamanthou[2], Roberto Tamassia[2], and Nikos Triandopoulos[3]

[1] Dept. of Computer Science, U. California, Irvine, USA
goodrich@ics.uci.edu
[2] Dept. of Computer Science, Brown University, USA
{cpap,rt}@cs.brown.edu
[3] Dept. of Computer Science, University of Aarhus, Denmark
nikos@daimi.au.dk

Abstract. We study the problem of authenticated storage, where we wish to construct protocols that allow to outsource any complex file system to an untrusted server and yet ensure the file-system's integrity. We introduce *Athos*, a new, platform-independent and user-transparent architecture for authenticated outsourced storage. Using light-weight cryptographic primitives and efficient data-structuring techniques, we design authentication schemes that allow a client to efficiently verify that the file system is fully consistent with the exact history of updates and queries requested by the client. In Athos, file-system operations are verified in time that is logarithmic in the size of the file system using optimal storage complexity—constant storage overhead at the client and asymptotically no extra overhead at the server. We provide a prototype implementation of Athos validating its performance and its authentication capabilities.

1 Introduction

Current trends in the design of data-storage systems are towards decentralized and networked architectures where data resides "in the cloud", outside any administrative control, and is being manipulated in storage units of minimal trust assumptions (e.g., NAS or SAN, storage providers, Internet-based computing). Operating on remotely managed data inherently entails security risks: when the storage provider is not trusted by the data source, verifying the integrity of the stored data and the correctness of the computations performed on this data is necessary to ensure the trustworthiness of the storage system; and verifying complex operations over general file systems efficiently is rather challenging.

* Research supported in part by the U.S. National Science Foundation under grants IIS–0713403, IIS-0713046, CNS-0312760 and OCI–0724806, the I3P Institute under a U.S. DHS award, the Center for Algorithmic Game Theory at the University of Aarhus under an award from the Carlsberg Foundation, the Center for Geometric Computing and the Kanellakis Fellowship at Brown University, and IAM Technology, Inc. The views in this paper do not necessarily reflect the views of the sponsors.

T.-C. Wu et al. (Eds.): ISC 2008, LNCS 5222, pp. 80–96, 2008.

In this paper, we study the problem of authenticating the integrity and operational correctness of a file system that is outsourced by a client to an untrusted server. We assume that the remote server's host machine and its storage units can behave maliciously. We wish to design authentication protocols that allow the client to efficiently verify the integrity of a dynamically evolving file system, namely to verify that its status is consistent with the exact history of file-system operations requested by the client, and to correctly detect any malicious data-update or data-retrieval patterns produced by the server. To conform to the outsourced data model, we require that the authentication protocols incur constant storage overhead at the client and asymptotically no extra storage costs at the server—otherwise, the client has no reason to outsource its data, at the first place—and also that verification is achieved efficiently, in time that is sublinear or logarithmic in the file system's size—or else, the client could trivially download the entire signed (and timestamped) file-system data on every operation.

Goals and Assumptions. Using cryptographic hashing is the state-of-the-art solution for verifying the integrity of simple put-get operations over a collection of files in the outsourced data model: the client locally keeps the hash of each file against which file retrievals or updates can be verified in constant time. The use of Merkle's tree [18] can reduce the client's space from linear to constant: the client only stores the root hash and both file retrievals and file updates (e.g., using existing techniques [3, 28]), can be verified in logarithmic time. Unfortunately, applying this approach in our setting provides only a partial solution: file-system integrity requires not only data integrity at the file or data-block level, but also integrity of the *directory hierarchy* of the file system. Indeed, all file-system operations are defined with respect to the directory path, and in many cases, the integrity of a file depends not only on its content, but also on its location in the file system. For example, the context of an .htaccess file depends on its location—its contents identify access policies, but its location is critical to identify the directories it protects. Our goal, therefore, is primarily to efficiently verify the directory hierarchy of the file system, and through this, any file-system or metadata operation that depends on this hierarchy. Of course, applying hashing over the directory tree (e.g., as in [7]), possibly augmented by the (balanced) hash trees that correspond to large files lying in the directory tree, can provide a space-optimal solution. However, this approach incurs linear verification and update costs, for the directory tree is unlikely to be balanced! Our goal is to design dynamic authentication schemes that overcome this problem.

Another solution is to have the client authenticate each file-system update it makes in the outsourced file system (e.g., using a signature or HMAC based on a private key), which has some major drawbacks, however. First, it allows for replay attacks, since determining the freshness of signed statements is difficult with such a scheme. Second, this solution requires the client to sign every possible path in the directory hierarchy in order to be able to authenticate locations. This can be especially inefficient, for example when the client performs a directory operation that moves a large directory to a new location. Another possibility is to assume that the outsourced file system is partially trustworthy (e.g., [21])

or a part of its architecture uses some tamper-resistant trusted hardware (e.g., using trusted computing platforms [27]). This assumption postulates that the networked file system is itself at least partially trusted, which is not that much different than simply trusting the hosting server in the first place. As we show in this paper, such trust is not necessary for the sake of efficiency or reliability.

In this work, we consider the outsourced data authentication problem in the single-client setting. However, in a multi-client setting, the problem of outsourced data authentication has drastically different characteristics. If communication between the clients is not allowed, a malicious server can easily perform an attack against data consistency: the server can effectively hide the most recent update on the data from a client requesting to read the data, by unrolling its state to the state the server had just before that update took place. Undetected without communication, this attack can be used to "fork" the view that this client has about the outsourced data, harming the consistency of the system. In this scenario, the best one can hope for is *fork-consistency* [16], effectively disallowing anything more than the forking attack, and various schemes securely achieve this property (e.g., [16, 20, 15, 4]). However, in the single-client setting the forking attack can be detected and prevented (e.g., by the hash-based solution), therefore, the fork-consistency property is no longer relevant in this setting. Although less general, the single-client model has its own merits. First, it naturally captures the security problems for a wide application area, where a single user outsources a personal file system to a storage provider. Second, in certain applications the multi-user setting is easily reduced to the single-client setting; for instance, in a networked file system, the client can simply abstract the OS kernel or a designated filer machine through which all file-system operation requests coming from many users are serialized to the untrusted remote storage devices. Third, in applications that can tolerate reasonable delays in the response time, and under reasonable assumptions about the availability of a *constant-size* shared trusted storage, the multi-client setting is also reduced to the single-client setting, achieving a stronger property than fork-consistency: conceptually the shared memory replaces the communication between parties.

Related Work. Previous work makes use of cryptographic hashing or signatures for primarily protecting the integrity of individual files or the corresponding data blocks that reside at storage units. Most of the systems (e.g., [5, 2, 9, 19]) provide file integrity using authentication information at the client that is proportional to the size n of the file system (i.e., the number of files or corresponding data blocks). More efficient constructions involve the use of Merkle trees [18] over the data blocks of individual files (e.g., [6, 24, 14, 21, 15]) or over the blocks or files of the entire file system (e.g., [8, 31]). Beyond hashing and signing, other space-efficient techniques have been proposed for file integrity, such as an entropy-based integrity method for encrypted (only) files [22] and a scheme based on the Galois counter mode [17], where however updates take linear time. Some constructions do authenticate the directory hierarchy or related meta-data of the file system, but, by hashing over the directory tree or signing each individual object, they result in linear update costs (e.g., [13, 7, 10]), or only support

verification of a limited set of operations (e.g., [15, 7, 10, 14]). Other schemes, additionally assume the existence of a trusted component at the untrusted server (e.g., [21, 25, 30], or some external trusted party (e.g., [31]) to authenticate file operations. Finally, SUNDR [15] and [16, 20, 4] use hashing and signatures to provide file-system integrity and fork-consistency in the multi-client setting; solving a harder problem, these schemes have increased performance costs.

Our Contributions. We present the design and a prototype implementation of an authentication architecture, which we call *Athos* (AuTHenticated Outsourced Storage), that supports an *authenticated outsourced file system* in the client-server model. We construct protocols for authenticating a rich set of file-system operations that are requested by the client and performed by the untrusted server. Our protocols support verification of the file system's full functionality by efficiently providing not only integrity of the stored data, but also integrity of the file-system directory structure. Security in our model corresponds to the natural notion of *consistency* in the single-client setting: at all times, the interaction with the server over any series of file-system operations should give the client the same view as the one obtained by a trusted server (as if the file system was never outsourced), and any deviation should be immediately detected. To achieve this, the client maintains only a hash digest of the file system, against which the validity of each operation performed by the server can be verified, using small proofs. These proofs are generated by an *authentication service* module that uses an authentication data structure stored in the server's untrusted memory, and runs in parallel with the file-system management module, and they consist of partial file system meta-data and hashes stored in the authentication data structure. This data structure defines the file-system digest, in a hash-tree fashion.

To achieve our efficiency goals, we use ideas from the domain of data authentication, employing efficient data structuring techniques for representing an entire file system. The challenge is to efficiently authenticate the directory hierarchy, which is typically highly unbalanced. We contribute two concrete authentication structures: Our first construction is based on a novel mapping of the directory hierarchy to a set of relations, and the authentication of put-get operations on this set using a skip list as the underlying authentication data structure. This approach achieves simplicity and low-cost authentication, and also leverages all the benefits of the widely researched authenticated dictionaries (e.g., [11]). Our second construction is of more theoretical interest, providing an optimal authentication scheme based on dynamic trees, a classical data structuring technique for operating on unbalanced trees in a balanced way. Overall, Athos achieves optimal storage usage (constant for the client and linear for the server) and efficient integrity verification (logarithmic or sublinear depending on the operation) and achieves generality by being agnostic of the specific implementation of the networked file system and by being platform-independent. Finally, a prototype implementation of Athos and an experimental evaluation of its verification capabilities for real-life file systems confirm our theoretical analysis.

Section 2 overviews our authentication model and Section 3 describes our authentication schemes. Section 4 presents the experimental evaluation of Athos

and discusses related issues. Section 5 presents our concluding remarks. Details on our construction that is based on dynamic trees and our experimental evaluation can be found in the Appendix. This extended abstract omits complete proofs and other details that will appear in the full version of the paper.

2 Model and Definitions

We study storage authentication in the following model (see also Figure 1). A client $\mathcal{C}$ owns and (incrementally) outsources a file system FS to an untrusted server $\mathcal{S}$. In additional to the file system, $\mathcal{S}$ hosts and controls an *authentication service* module $\mathcal{A}$ that stores authentication information about FS. The file system is generated and queried through a series of *update* and *query* operations issued by the client $\mathcal{C}$. At any time, $\mathcal{C}$ keeps some *state information* s that encodes information about the current state of FS. If $\mathcal{P}$ is the set of operations supported over the file system, then the communication protocol is as follows:

1. Client $\mathcal{C}$ keeps state information s and issues a query or update operation $o \in \mathcal{P}$ to the server $\mathcal{S}$.
2. Server $\mathcal{S}$ runs a certification algorithm, which performs operation o and accordingly answers the query or updates FS to a new version FS', and, by using $\mathcal{A}$, also generates a *verification* or respectively *consistency proof* π which is returned to $\mathcal{C}$, along with the result ρ of the operation; ρ is the corresponding answer if o is a query operation or else the empty string $\perp$. We write $\pi \leftarrow \mathsf{certify}(o, FS, FS', \rho)$.
3. Client $\mathcal{C}$ runs a verification algorithm, which takes as input the current state s, the operation o along with its result ρ, and the corresponding (consistency or verification) proof π and either accepts or rejects the input. If the input is accepted the state s is appropriately updated to state s', where $s' = s$ if o is a query operation or else $s' \neq s$. We write $\{(yes, s'), (no, \perp)\} \leftarrow \mathsf{verify}(s, \rho, \pi)$.

We call the pair of algorithms ($\mathsf{certify}$, verify) an *authenticated storage scheme*.[1] The security property we wish such a scheme to satisfy expresses the intuitive requirement that the verification performed at $\mathcal{C}$ must be a reliable test for checking the file system's integrity. Let $\mathsf{operate}(\cdot, \cdot)$ be the algorithm that, given the current file system FS and an operation $o \in \mathcal{P}$, performs o and updates FS to FS'. We write $(FS', \rho) \leftarrow \mathsf{operate}(o, FS)$ ($\rho = \perp$ for updates and $FS' = FS$ for queries). We say that state s is *consistent* with FS_τ for a series τ of operations on FS, if s and FS_τ have been computed by running algorithms $\mathsf{operate}$, $\mathsf{certify}$ and verify sequentially for all operations in series τ starting from FS.

Definition 1 (Security of authenticated storage schemes.). *We say that an authenticated storage scheme* ($\mathsf{certify}, \mathsf{verify}$) *(with security parameter κ) is* secure*, if for any series of operations τ and a state s that is consistent with file system FS_τ for τ on an initially empty file system, the following conditions hold:*

[1] Both algorithms take as input also a public key that is known by both $\mathcal{C}$ and $\mathcal{S}$.

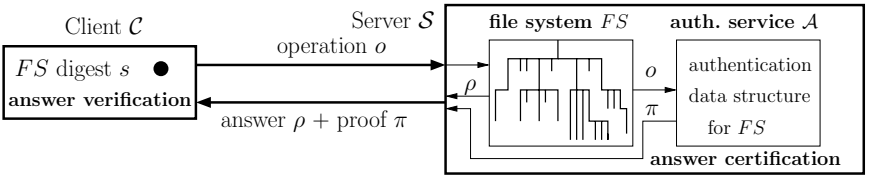


Fig. 1. The authenticated data storage model. Keeping only constant-size state s, client $\mathcal{C}$ remotely manages a file system FS that resides at untrusted server $\mathcal{S}$. Every query or update operation o requested by $\mathcal{C}$ on FS is certified by $\mathcal{S}$, using an authentication service module $\mathcal{A}$ (that stores authentication information related to FS) to produce a verification or consistency proof π; this proof is used by $\mathcal{C}$, along with the result ρ of the operation, to verify that the request was handled consistently, and finally update s.

Correctness. *For any $o \in \mathcal{P}$, when $(FS'_\tau, \rho) \leftarrow \mathsf{operate}(o, FS_\tau)$, it holds that $(yes, s') \leftarrow \mathsf{verify}(s, \rho, \mathsf{certify}(o, FS_\tau, FS'_\tau, \rho))$. I.e., for any correctly performed operation,* certify *generates a proof that is always accepted by* verify*, which also computes a new, consistent with the new file system FS'_τ, state s'.*

Consistency. *For any series τ of operations and new operation o, such that state s is consistent with file system FS_τ for τ on an initially empty file system and $(FS'_\tau, \rho) \leftarrow \mathsf{operate}(o, FS_\tau)$, then for any polynomial-time adversary, controlling $\mathcal{S}$ and having oracle-access to algorithm* verify*, that on input the file system FS_τ, series τ and operation o, produces proof π' and result ρ', whenever $(yes, s') \leftarrow \mathsf{verify}(s, \rho', \pi')$, then the probability that either $\rho' \neq \rho$ or s' is not consistent with FS'_τ for operation o on FS_τ is negligible (in the security parameter κ). I.e., assuming a polynomially bounded adversary that observes polynomially many protocol invocations and then produces a pair of proof π' and result ρ', if ρ' and π' for the new operation o are accepted by* verify*, then for all but negligible probability the operation has been performed correctly and the new state is consistent with the new file system.*

Starting from an initially empty set and using a secure authenticated storage scheme and the appropriate series of updates, client $\mathcal{C}$ is able to "export" *any* file system to server $\mathcal{S}$, such that $\mathcal{C}$ has a consistent state with the current file system. Therefore, the file system is consistent with the history of updates and all future operations will be verified. With respect to efficiency, we say that an authenticated storage scheme is time-*efficient* if the verification time at $\mathcal{C}$ is sublinear in the file-system size, and space-*optimal* if $\mathcal{C}$ stores state of constant size. We next exhibit *time-efficient*, *space-optimal* and *secure* authenticated storage schemes for a rich set of operations on an outsourced file system.

3 Efficient Authenticated Storage

To give some intuition of our general approach, let us consider the special case where we want to implement an *authenticated map* in the client-server outsourced

storage model; actually, this authentication functionality on the map data structure will also be our core authentication tool for verifying file system operations. Each entry of the map is a tuple (k, v), where v is a value corresponding to a key k (v can be a collection of objects). The entries of the map are sorted according to their keys (using some comparator). The authenticated map data structure resides at the server. Using a *hashing scheme* for skip lists, i.e., a hierarchical way to produce a hash digest by recursively applying a cryptographic hash function h over some data (see, e.g., [11]), we can define a digest of the authenticated map, computed according to the skip-list tree structure (Figure 2(a)).

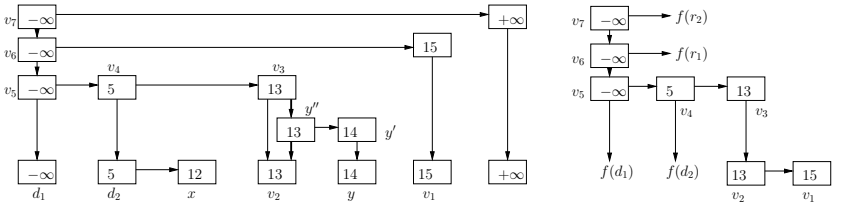


Fig. 2. (a) The skip list hashing scheme for verifying operations on a map data structure and insertion of key 14. (b) The consistency proof P returned by $\mathcal{S}$, containing all the hashing and structural information needed to verify the consistency of P subject to the current digest and to locally perform the update.

Let d_0 be the initial digest stored at the client $\mathcal{C}$ which is consistent with the current state of the skip list. Suppose now that $\mathcal{C}$ wants to insert a new key x. The server $\mathcal{S}$ returns to $\mathcal{C}$ a consistency proof that consists of the search path P in the unsuccessful search for x before the update (Figure 2(b)). Path P is related with the key insertion, satisfying the following properties, which in turn imply the security of the scheme. First, P contains the two keys, say $\mathsf{succ}(x)$ and $\mathsf{pred}(x)$, that are the successor and predecessor of x in the ordering of the keys, and also contains all the necessary *hashing* information (hash values) that allows $\mathcal{C}$ to recompute the current digest d_0 starting from $\mathsf{succ}(x)$ and $\mathsf{pred}(x)$ and hashing according to the hashing scheme that is used. Due to the collision-resistance property of the hash function, $\mathcal{C}$ can check if the received path is the correct one, and if P is verified, $\mathcal{C}$ verifies that key x is not in the directory. Also, $\mathcal{C}$ knows the position at which this file should be added. Second, P contains all the necessary *structural* information that allows $\mathcal{C}$ to locally perform the update in the hashing scheme that corresponds after the file insertion, by placing x between $\mathsf{succ}(x)$ and $\mathsf{pred}(x)$ and computing the new hash values for only those nodes of the skip list that need a new hash. Knowing the new hash values, $\mathcal{C}$ can compute the new digest d_0', which is consistent with the update. Thus, the key insertion (performed by $\mathcal{S}$) can be verified in two steps: first, path P is verified and then it is used to locally perform the update and compute the new digest. Using results on authenticated skip lists (e.g., [23, 11]) we have the following:

Lemma 1. *There exists an authenticated storage scheme for operations on n key-value pairs in a map that is based on an authenticated skip list, with the*

following expected complexity bounds: (i) The expected update (insertion and removal), query and verification time is $O(\log n)$ w.h.p.; (ii) The expected size of the consistency and verification proof (communication cost) is $O(\log n)$ w.h.p.

Here, *update time* is the time required by $\mathcal{S}$ to do the actual update, *query time* is the time $\mathcal{S}$ needs to compute the (consistency or verification) proof, *verification time* is the time that $\mathcal{C}$ needs in order to process the proof and validate or reject the query or the update. Note that for *set-membership queries and updates* (through which we are going to implement all file system operations) the size of a proof is always asymptotically equal to the verification time; therefore, the verification time bounds will indirectly imply the size of the proof.

We next use the authenticated-map functionality to verify more complex operations on general file systems. Let T be the tree that corresponds to the hierarchy induced by the structure of the directories and files of a file system, where the left-right ordering of sibling nodes coincides with the chronological order of the node creations. The idea is to carefully map T's structural information to a set of special entries and store this set in an authenticated map, in a way that allows to authenticate the integrity of the entire file system. Node v in T (a directory or file) defines an authenticated-map entry that stores, under key $\mathsf{key}(v)$ that is the corresponding i-node in the file-system, the following fields:

- name: the actual name of the node of the file system;
- file: a hash (e.g., SHA-1) of the file represented by v (null for a directory);
- $\mathsf{key}(\mathsf{parent})$: the key of the entry corresponding to the parent node of v;
- $\mathsf{key}(\mathsf{sibling})$: the key of the entry corresponding to the successor sibling of v in T (null if v is the last created node of the children list);
- $\mathsf{key}(\mathsf{backsibling})$: the key of the entry corresponding to the predecessor sibling of v in T (null if v is the first created node of the children list);
- $\mathsf{key}(\mathsf{child})$: the key of the entry corresponding to the first created child of v.

We next map each file-system query or update to a small set of (regular) query or update operations in the authenticated map, effectively reducing file-system operations to set-membership operations. We have the following:

Theorem 1. *Assuming the existence of collision-resistant hash functions, there exists a secure and space-optimal authenticated storage scheme that is implemented with skip lists, achieving the following performance, where n is the size of the file-system tree T, T_v is the subtree rooted on node v, ℓ_v is the number of children of node v and $\Pi = \pi_1\pi_2\ldots\pi_k$ is a path in T: (1) The authentication of any path Π takes $t(\Pi) = O(k \log n)$ query and verification time; (2) Query operations $\mathsf{cd}(\Pi)$, $\mathsf{read}(\Pi)$ and update operations, $\mathsf{write}(\Pi)$, $\mathsf{rm}(\Pi)$, $\mathsf{mkdir}(\Pi)$, $\mathsf{touch}(\Pi)$ take $t(\Pi)$ query, verification and update, query, verification time respectively; (3) Query operation $\mathsf{ls}(\Pi)$ takes $t(\Pi) + O(\ell_{\pi_k} \log n)$ query and verification time; (4) Update operation $\mathsf{rmdir}(\Pi)$ takes $t(\Pi) + O(|T_{\pi_k}| \log n)$ update, query and verification time; (5) Update operation $\mathsf{mv}(\Pi, \Pi')$ takes $t(\Pi) + t(\Pi')$ update, query and verification time.*

We now discuss another possible method of representing the file system using a skip list. Instead of setting the i-node number as node's v key, we can set as $\mathsf{key}(v)$

Table 1. Efficiency comparison of our authenticated storage schemes w.r.t. the query, update and verification times, using skip lists (local and global approaches) and dynamic trees. Here, n is the size of the file system, $\Pi = \pi_1\pi_2\ldots\pi_k$ is the directory argument, ℓ is the size of the children list and T is the subtree rooted on π_k.

operation	skip list (local)	skip list (global)	dynamic tree
$\mathsf{cd}(\Pi)$, $\mathsf{touch}(\Pi)$, $\mathsf{read}(\Pi)$ $\mathsf{write}(\Pi)$, $\mathsf{rm}(\Pi)$, $\mathsf{mkdir}(\Pi)$	$O(k \log n)$	$O(\log n + k)$	$O(\log n + k)$
$\mathsf{ls}(\Pi)$	$O((k+\ell)\log n)$	$O(\ell(\log n + k))$	$O(k + \ell + \log n)$
$\mathsf{rmdir}(\Pi)$	$O((k+\lvert T\rvert)\log n)$	$O(\lvert T\rvert \log n + k)$	$O(k + \log n)$
$\mathsf{mv}(\Pi, \Pi')$	$O((k+k')\log n)$	$O(\lvert T\rvert \log n + k + k')$	$O(k + k' + \log n)$

the name of the path from the file-system root to node v (e.g., the key for file pub.txt lying in path /users/user/ is now the string "/users/user/pub.txt"). Thus, in the previous representation we stored "local" information, whereas now we rather store "global" information. This solution yields better complexity bounds for the path authentication (which is now $t(\Pi) = O(\log n + |\Pi|)$). However, update operation $\mathsf{mv}(\Pi, \Pi')$ takes $O\left(t(\Pi) + t(\Pi') + |T| \log n\right)$ update, query, and verification time, where T is the subtree rooted at $\pi_{|\Pi|}$. This representation is suitable for cases where the majority of the operations are file system navigations and move operations are less frequent. Finally, by using authenticated path operations [12] implemented with dynamic trees [26], we get the following:

Theorem 2. *Assuming the existence of collision-resistant hash functions, there exists a secure, time-efficient and space-optimal authenticated storage scheme that is implemented with dynamic trees, achieving the following performance, where n is the size of the file system tree and ℓ_v is the number of children of v: (1) The authentication of any path Π takes $t(\Pi) = O(k + \log n)$ query and verification time; (2) Query operations $\mathsf{cd}(\Pi)$, $\mathsf{read}(\Pi)$ and update operations $\mathsf{write}(\Pi)$, $\mathsf{rm}(\Pi)$, $\mathsf{mkdir}(\Pi)$, $\mathsf{touch}(\Pi)$ take $t(\Pi)$ query, verification and update, query, verification time respectively; (3) Query operation $\mathsf{ls}(\Pi)$ takes $t(\Pi) + O(\ell_{\pi_k} + \log n)$ query and verification time; (4) Update operation $\mathsf{rmdir}(\Pi)$ takes $t(\Pi)$ update, query and verification time; (5) Update operation $\mathsf{mv}(\Pi, \Pi')$ takes $t(\Pi) + t(\Pi') + O(\log n)$ update, query and verification time.*

A more detailed description of this scheme appears in the Appendix. Table 1 presents a comparison between the efficiency levels achieved by our schemes.

Security. Our authenticated storage schemes are based on the following general approach. Given a secure hashing scheme $\mathcal{H}$ for a specific query type $\mathcal{Q}$, that is, a directed acyclic graph that defines how a hash digest is computed from a data set and a corresponding authentication structure,[2] we augment $\mathcal{H}$ to a new hashing scheme $\mathcal{H}'$ that additionally encodes (in its produced digest) the entire structural and balancing information that is defined in the underlying authentication

[2] Against this (authentic) digest answers to queries in $\mathcal{Q}$ can be efficiently verified; this is the general verification technique used by authenticated data structures.

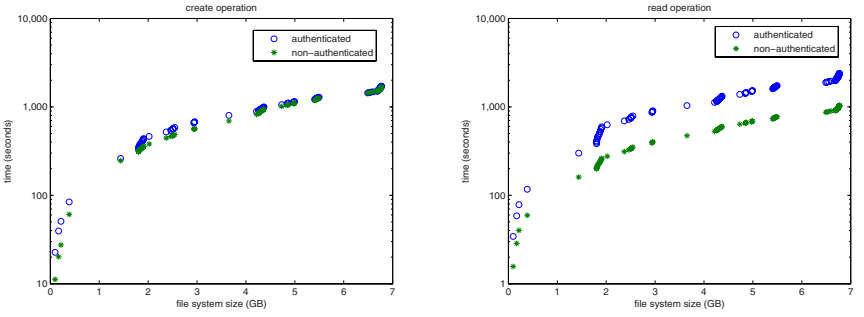
structure. In particular, if the hash value h_v of node v in the data structure is computed as $h_v = h(h_{u_1}, \ldots, h_{u_k})$ in $\mathcal{H}$, we define $h_v = h(h_{u_1}, \ldots, h_{u_k}, h(b_v, s_v))$ in $\mathcal{H}'$, where b_v, s_v describe the balancing and respectively structural information about node v. In our constructions, we make use of the hashing schemes corresponding to the skip list and the dynamic-tree data structure for efficiently verifying set-membership [11] and respectively path property [12] queries.

Given the augmented hashing trees, security is proved as follows. Starting from the state corresponding to the empty file system, we inductively show that after any update on the file system the client $\mathcal{C}$ updates its state s consistently for the new update on the currently existing file system FS. For both data structures used in our schemes, the consistency proof by the definition of the corresponding augmented hashing scheme $\mathcal{H}'$ contains all the balancing and structural information that completely characterizes the changes in FS due to the update. Assuming that the state is consistent, the consistency proof coming from an honest server $\mathcal{S}$ will be verified, thus also the balancing and structural information related to the update. Thus, $\mathcal{C}$ is able to locally perform the correct update as if $\mathcal{C}$ had direct access to the entire file system FS, thus is able to correctly and consistently update his state s to s', which is simply the new digest according to $\mathcal{H}'$. Given this invariant, any query is securely verified since the underlying hashing scheme is secure: assuming that finding hash collisions is computationally hard, any malicious behavior by $\mathcal{S}$ will be rejected by the verification algorithm, since any undetected inconsistency corresponds to a collision.

4 Analysis, Experiments and Discussion

We have developed a prototype implementation of Athos using skip lists. Our implementation uses a flat representation of the file system tree since this representation outperforms dynamic trees when the depth of the tree is less than 200 [29], which typically occurs in file systems. We have implemented the authentication service (both the server and the client) in Java. The experiments were conducted on a 64-bit, 2.8GHz Intel based, dual-core, dual processor machine with 2GB main memory and 2MB cache, running Debian Linux 3.1 with Linux kernel 2.6.15 and using the Sun Java JDK 1.5. The time consumed by the garbage collector is excluded from the presented times. We have implemented authenticated versions of all the major commands of a file system (all the commands described in Theorem 1) and the basic functionality of a skip list. We executed the experiments on a remote file system (that lies however in close proximity to the terminal machine), the tree of which consists of roughly 77,779 nodes, of which 61,241 are files and the rest are directories. The average size of the files is 1.22 MB. The total size of the file system is 6.92 GB. However, the distribution of the files is not uniform (certain subtrees are very "heavy").

In Figures 3(a)/3(b) we plot the time taken to write/read our test file system as a function of the size of the portion of the file system processed. Note that our authentication service does not add much overhead to the non-authenticated write/read. Also note that the overhead of the authentication service is more



(a) Writing the file system (b) Reading the file system

Fig. 3. Times to write and to read our test file system, using standard and authenticated operations. The cumulative time elapsed is plotted as a function of the size of the portion of file system processed. Each point corresponds to a new batch of 1,000 files processed. Since files have different sizes, the points on the plot are not uniformly spaced in the horizontal direction.

noticeable in the read experiment, with an average overhead per node (directory or file) of 17.61 ms. This is due to the fact that, when we read a file x, that lies in a path Π, we have to authenticate *both* the contents of the file and the existence of the path (by Theorem 1, this task takes $O(|\Pi| \log n)$ time). Also, for a directory d, we have to issue $|\mathsf{children}(d)|$ queries to the skip list in order to authenticate completeness. Due to space limitations, more experimental results can be found in the Appendix.

Discussion. Our protocols are designed in the client-server model. However, certain applications that require file-system integrity may involve a large number of users, and therefore, to achieve full consistency user interaction is necessary,[3] which results in impractical protocols (since, without other assumptions, n users need to exchange $\Omega(n)$ messages after any update). Unless one resorts to fork-consistency [16, 4], some communication assumptions must be made. For instance, when different users access a remote file system through the same network infrastructure, our protocols are applicable if we assume a single *designated* trusted client that serializes all users' operations and verifies them locally.[4]

An additional issue is related to failure recovery and persistent in a real-life usage of our authentication protocols. In the case of an unsuccessful verification

[3] To see why, assume any secure protocol for verifying the integrity of outsourced storage, and consider an update on the data performed and verified by user A. Consider the next operation on the data issued by user B. Without interaction, even if users locally keep unbounded state, replay attacks are impossible to defeat, since a malicious server can ignore A's updates on the data without being noticed by B.

[4] That is, Athos' verification client can serve as an add-on module of the hosting operating-system kernel that runs in parallel with the system's filer.

of a file system operation, Athos can provide to the higher (or hosting) application complete information about the problematic operation and the current state of the file system in terms of its integrity. In particular, Athos functionality can characterize the exact location in the file system where integrity was not verified and thus pinpoint which file or directory was maliciously (or accidentally) modified by the untrusted server or by the remote storage devices. By keeping appropriate additional information, the higher application is thus able to infer useful information for failure recovery and a complete view of the problem. For instance, one can find which concrete user and with which concrete operation most recently, correctly accessed the (currently problematic) file or directory. Additionally, by using our skip list based authentication approach in combination with existing techniques [1] for authenticating membership queries in the past (i.e., queries that span through previous states of a data set), Athos can offer persistent authentication capabilities, where file-system operations or queries about past views of the file system can be issued and authenticated. In this way, we can support secure audit of the entire outsourced file system.

Finally, Athos can also support authentication of files at the block level. To do that, we introduce one more level of authentication using a skip list on top of a file. The digest of this skip list is now what is stored in the original skip list. The client can update individual blocks of the file and also query for certain blocks of the file. The length of the proof depends on the granularity we use to partition the file into blocks. Obviously there is a trade-off between the size of the verification proof and the data someone needs to download for authentication.

5 Conclusions

In this paper we present efficient protocols for verifying the integrity of a file system that is outsourced to an untrusted storage facility. We use cryptographic hashing and efficient data structures to produce and incrementally update, after file system operations, a short and secure digest of the entire file system. This digest is used by a client to efficiently verify that the file system is fully consistent with the history of query and update operations requested by the client to the host server. Our protocols authenticate both the contents of the files and the directory hierarchy of the file system, thus verifying a rich set of file system operations. The authentication of operations uses a short verification or consistency proof that is computed by the server and involves communication and computation overheads that are sublinear in file system size. This makes our authentication schemes applicable in settings where low-computing power and/or low-storage devices need to access a remote file system in a secure way. We authenticate common and important file system operations such as `cd`, `ls` in *logarithmic* time and, through a prototype implementation, we experimentally confirm the efficiency and practicality of our authentication methods.

References

[1] Anagnostopoulos, A., Goodrich, M.T., Tamassia, R.: Persistent authenticated dictionaries and their applications. In: Proc. Information Security Conference, pp. 379–393 (2001)
[2] Blaze, M.: A cryptographic file system for Unix. In: Proc. Conference on Computer and Communications Security, pp. 9–16 (1993)
[3] Blum, M., Evans, W., Gemmell, P., Kannan, S., Naor, M.: Checking the correctness of memories. In: Proc. Foundations of Comp. Science, pp. 90–99 (1991)
[4] Cachin, C., Shelat, A., Shraer, A.: Efficient fork-linearizable access to untrusted shared memory. In: Proc. Principles of Distr. Computing, pp. 129–138 (2007)
[5] Cattaneo, G., Catuogno, L., Sorbo, A.D., Persiano, P.: The design and implementation of a transparent cryptographic file system for Unix. In: Proc. USENIX Annual Technical Conference, pp. 199–212 (2001)
[6] Fu, K.: Group sharing and random access in cryptographic storage file systems. Master's thesis, Massachusetts Institute of Technology (May 1999)
[7] Fu, K., Kaashoek, M.F., Mazières, D.: Fast and secure distributed read-only file system. ACM Trans. Comput. Syst. 20(1), 1–24 (2002)
[8] Fujita, T., Ogawara, M.: Arbre: A file system for untrusted remote block-level storage. IPSJ Digital Courier 1, 381–393 (2005)
[9] Gobioff, H., Nagle, D., Gibson, G.A.: Integrity and performance in network attached storage. In: Proc. International Symposium on High Performance Computing, pp. 244–256 (1999)
[10] Goh, E.-J., Shacham, H., Modadugu, N., Boneh, D.: SiRiUS: Securing Remote Untrusted Storage. In: Proc. Network and Distr. Sys. Security, pp. 131–145 (2003)
[11] Goodrich, M.T., Tamassia, R., Schwerin, A.: Implementation of an authenticated dictionary with skip lists and commutative hashing. In: Proc. DARPA Information Survivability Conference and Exposition, pp. 68–82 (2001)
[12] Goodrich, M.T., Tamassia, R., Triandopoulos, N., Cohen, R.: Authenticated data structures for graph and geometric searching. In: Proc. RSA Conference—Cryptographers' Track, pp. 295–313 (2003)
[13] Jammalamadaka, R.C., Gamboni, R., Mehrotra, S., Seamons, K.E., Venkatasubramanian, N.: gVault: A gmail based cryptographic network file system. In: Proc. Conf. on Data and Applications Security, pp. 161–176 (2007)
[14] Kallahalla, M., Riedel, E., Swaminathan, R., Wang, Q., Fu, K.: Plutus: Scalable secure file sharing on untrusted storage. In: Proc. USENIX Conference on File and Storage Technologies, pp. 29–42 (2003)
[15] Li, J., Krohn, M.N., Mazières, D., Shasha, D.: Secure untrusted data repository (SUNDR. In: Proc. Operating Systems Design and Impl., pp. 121–136 (2004)
[16] Mazières, D., Shasha, D.: Building secure file systems out of byantine storage. In: Proc. Principles of Distributed Computing, pp. 108–117 (2002)
[17] McGrew, D.: Efficient authentication of large, dynamic data sets using galois/counter mode. In: Proc. Security in Storage Workshop, pp. 89–94 (2005)
[18] Merkle, R.C.: A certified digital signature. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 218–238. Springer, Heidelberg (1990)
[19] Miller, E.L., Long, D.D.E., Freeman, W.E., Reed, B.: Strong security for network-attached storage. In: Proc. File and Storage Tech., pp. 1–13 (2002)
[20] Oprea, A., Reiter, M.K.: On consistency of encrypted files. In: Dolev, S. (ed.) Proc. International Symposium on Distributed Computing, pp. 254–268 (2006)

[21] Oprea, A., Reiter, M.K.: Integrity checking in cryprographic file systems with constant trusted storage. In: Proc. USENIX Security, pp. 183–198 (2007)
[22] Oprea, A., Reiter, M.K., Yang, K.: Space-efficient block storage integrity. In: Proc. Network and Distributed System Security Symposium, pp. 17–28 (2005)
[23] Papamanthou, C., Tamassia, R.: Time and space efficient algorithms for two-party authenticated data structures. In: Proc. Information and Communications Security, pp. 1–15 (2007)
[24] Pletka, R., Cachin, C.: Cryptographic security for a high-performance distributed file system. In: Proc. Mass Storage Systems Tech., pp. 227–232 (2007)
[25] Sarmenta, L.F.G., van Dijk, M., O'Donnell, C.W., Rhodes, J., Devadas, S.: Virtual monotonic counters and count-limited objects using a TPM without a trusted OS. In: Proc. Workshop on Scalable Trusted Computing, pp. 27–41 (2006)
[26] Sleator, D.D., Tarjan, R.E.: A data structure for dynamic trees. J. Comput. Syst. Sci. 26(3), 362–381 (1983)
[27] Smith, S.W.: Trusted Computing Platforms: Design and Applications. Springer, Heidelberg (2005)
[28] Tamassia, R., Triandopoulos, N.: Efficient content authentication in P2P networks. In: Proc. Applied Cryptography and Network Security, pp. 354–372 (2007)
[29] Tarjan, R., Werneck, R.: Dynamic trees in practice. In: Proc. Workshop on Experimental Algorithms, pp. 80–93 (2007)
[30] van Dijk, M., Rhodes, J., Sarmenta, L.F.G., Devadas, S.: Offline untrusted storage with immediate detection of forking and replay attacks. In: Proc. Workshop on Scalable Trusted Computing, pp. 41–48 (2007)
[31] Yumerefendi, A.Y., Chase, J.S.: Strong accountability for network storage. In: Proc. Conference on File and Storage Tech., pp. 77–92 (2007)

Appendix

Dynamic Trees Implementation. Let T be the tree that represents our file system. The leaves of T are either files or empty directories. We transform T to a new data structure which is essentially a tree $\mathcal{T}$ of paths (Figure 4(b)). Our data structure is based on dynamic trees [26]. On the tree $\mathcal{T}$, we make use of the hashing scheme for authentication of *path properties* in trees from [12]. This hashing scheme is defined over trees of the form of our final tree $\mathcal{T}$ and allows authentication of path properties that satisfy the concatenation criterion: Let $p = p' \| p''$ be a path in T that is the concatenation of paths p' and p''. A

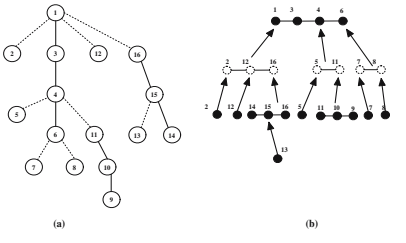


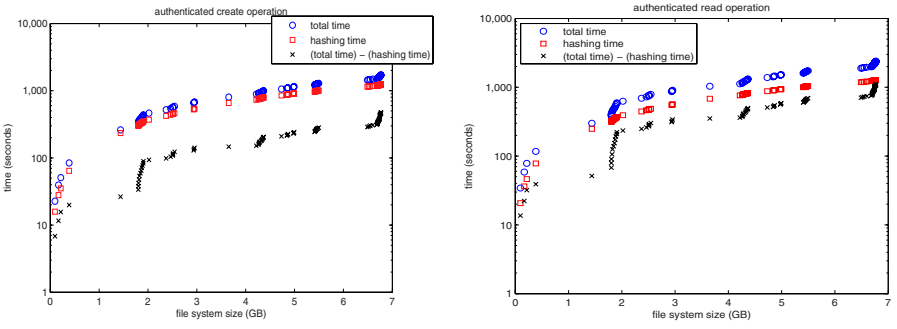
Fig. 4. (a) File system tree. (b) The tree of paths $\mathcal{T}$.

path property $\mathcal{P}$ satisfies the *concatenation criterion* if $\mathcal{P}(p) = \mathcal{F}(\mathcal{P}(p'), \mathcal{P}(p''))$, where $\mathcal{F}$ is a function that can be computed in $O(1)$ time; e.g., a path property that satisfies this property is the *length of the path*, where $\mathcal{F}$ is "addition".

We extend this hashing scheme to authenticate path properties not only for paths in the original tree T but also for dashed paths in the intermediate tree $\mathcal{T}$ (i.e., properties of paths related to siblings). This extension is performed by including in the hashing scheme information associated with the files and subdirectories of any directory. Also, we include in the dashed path $d(v)$ related to v, the node in T (file or subdirectory) that corresponds to the solid child of v in T (so that no file is missed). Finally, we augment the hashing scheme to include structural and balancing information related to T: now the hash value of any node in $\mathcal{T}$ includes its sibling rank and weight.

We now relate operations of the file system with certain path properties in T or dashed-path properties. In order to do this, we define the appropriate path properties of interest: every node v of the tree is related with a constant-size set of node attributes $\{N_1(v), \ldots, N_k(v)\}$. These for example can be the weight of v or other variables that we want to relate with this node. We call the set of these node attributes the *node property* $\mathcal{N}(v)$ of this node. For the case of the file system, we define the node property $\mathcal{N}(v)$ of a node v to contain two attributes: $S(v)$ and $C(v)$. $S(v)$ is the name of the file or directory and $C(v)$ is the hash of the certain file or directory. If node v represents a directory, we define $C(v) = \{\emptyset\}$, otherwise $C(v)$ is a hash of the corresponding to v file. Similarly, every path p is related with a set of path attributes $\{P_1(p), \ldots, P_k(p)\}$. These can be the *length* of a path or other variables that we want to relate with this path. We call the set of these path attributes the *path property* $\mathcal{P}(p)$ of this path. The path attributes can be defined as a function of the corresponding node attributes. In our case, we define the first path attribute $S(p)$ of a path $p = u_1, \ldots, u_\ell$ as $S(p) = \bigotimes_{i=1}^{\ell} S(u_i)$. This is actually the name of the path ($\otimes$ denotes "string concatenation"). The second path attribute is similarly defined to be the content of the path $C(p)$. Hence, $C(p) = \bigoplus_{i=1}^{\ell} C(u_i)$, where $\oplus$ is simply the union operator. Note that the content of a path that consists only of directories is empty. Also the path property $\mathcal{P}(p) = (S(p), C(p))$ for any path $p = p'|p''$ of the file system satisfies the concatenation criterion since $S(p) = S(p') \otimes S(p'')$ and $C(p) = C(p') \oplus C(p'')$. Hence, in the file system context, we can authenticate the major file system operations (see Theorem 2), by reducing them to an appropriately path property query, where we also use complexity analysis in [12]. We finally note that the consistency proof used by the client to do the updates has logarithmic size: since all the update operations described above take logarithmic time, they cannot visit more than $O(\log n + |\Pi|)$ nodes of the tree. Hence, the server can send structural and hashing information of size $O(\log n + |\Pi|)$ that allows the client to update the digest.

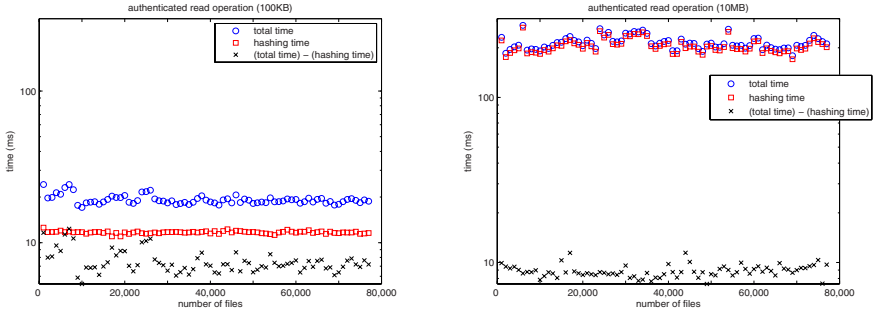
Additional Experimental Results. In Figure 5, we further analyze the time to read and write the test file system with authentication by accounting separately for the time taken to perform hashing (these experiments are a more fine-grained analysis of the experiments we presented before). We can see that for both the read and write experiments, the hashing time dominates the



(a) Authenticated create operation (b) Authenticated read operation

Fig. 5. Cumulative times for separate parts of the authenticated operations create/read. The most expensive part of either an authenticated create or an authenticated read is the hashing time, as indicated in the above figures. The remaining time is the time needed to send over data to the skip list.

computation. When writing the file system, we need to hash each file and then store the hash in the skip list, whereas to read the file system, we need to hash what we are reading in order to compare it with the authenticated hash that is returned by the skip list and was stored there during creation. We also note that the interaction with the authentication service ((total time) − (hashing time)) increases when our program parses a "light" region of the file system (e.g., the region around 2GB in Figure 5(b)). This is due to the fact that more files are being processed in less amount of time (the "light" region does not contain large files) and therefore the communication with the authenticated skip list increases. Finally, we observe that on average, hashing accounts for 73% of the write time and for 53% of the read time. This overhead is necessary in any authentication method based on cryptographic hashing.



(a) Authenticated read (100KB file) (b) Authenticated read (10MB file)

Fig. 6. Average time of an authenticated read operation. Every point is the average time (over 100 executions) for reading a file of certain size in a file system of varying size. The larger the file, the smaller is the difference (total time) − (hashing time).

In Figures 6(a) and 6(b), we plot the average time for reading a file with authentication, as a function of the number of nodes in the file system. Each point $p(x)$ is the average of 100 authenticated reads on a file system that contains x files. Note that these plots are not cumulative. For a 100KB file, the hashing time is about half the total time, whereas for a 10MB file, the hashing time is almost equal to the total time.

BotTracer: Execution-Based Bot-Like Malware Detection

Lei Liu[1], Songqing Chen[1], Guanhua Yan[2], and Zhao Zhang[3]

[1]Dept. of Computer Science, George Mason University
{lliu3,sqchen}@cs.gmu.edu
[2]Information Sciences, Los Alamos National Lab
ghyan@lanl.gov
[3]Dept. of Electrical and Computer Engineering,
Iowa State University
zzhang@iastate.edu

Abstract. Bot-like malware has posed an immense threat to computer security. Bot detection is still a challenging task since bot developers are continuously adopting advanced techniques to make bots more stealthy. A typical bot exhibits three invariant features along its onset: (1) the startup of a bot is automatic without requiring any user actions; (2) a bot must establish a command and control channel with its botmaster; and (3) a bot will perform local or remote attacks sooner or later. These invariants indicate three indispensable phases (startup, preparation, and attack) for a bot attack. In this paper, we propose BotTracer to detect these three phases with the assistance of virtual machine techniques. To validate BotTracer, we implement a prototype of BotTracer based on VMware and Windows XP Professional. The results show that BotTracer has successfully detected all the bots in the experiments without any false negatives.

Keywords: Botnet, malware detection, virtual machine.

1 Introduction

Bots and botnets have become one of the most serious threats to Internet security in recent years [14] [22]. Compared with other malware like virus and worms, bot behavior can be very stealthy, making their detection extremely difficult. For example, a bot can stay inactive without any dramatic activities for a long time. Oftentimes, a bot generates only a small amount of traffic, which is hidden among legitimate traffic. Some of botnet research has focused on the understanding of bots and botnets. For example, Barford et al. have analyzed in-depth bot source code [9] and provided insights from several perspectives, while in [27], through trace collection and analysis, authors observed the real-world botnet behavior. Dagon et al. have studied the botnet propagation using time zones [15]. Some research [25] has studied how to identify non-human behavior characteristics in traffic and build IRC server scanners to identify potential botnets. To counter

T.-C. Wu et al. (Eds.): ISC 2008, LNCS 5222, pp. 97–113, 2008.

botnets, honeypots have been used to infiltrate the command and control network of botnets [17].

While researchers are improving the detection and defense schemes, bot developers are also constantly making bots more stealthy. Due to the scrutiny on IRC channels, today's bots are bound to other popular applications (e.g., Web browsers [6]) or protocols (e.g., HTTP [16]). Distributed P2P-based botnets, which are much more difficult to detect and shut down than centralized botnet architectures, have also been developed [19] [21] [28] [30]. Advanced bots like `Spam-Mailbot` [12] have applied encryption to defeat traffic scanning.

Recent improvements on bot techniques call for better bot detection techniques. The distinguishing feature of a typical bot attack is three indispensable phases:

- *Automatic startup:* Different from those virus or worms (e.g., email worms) that rely on user intervention, a bot can be started automatically by modifying the automatic startup process list or registry entries. This is essential for the bot to actively initialize the command and control channel with the botmaster in order to receive commands.
- *Command and control channel establishment:* In spite of various bot hiding techniques, existing bots all need to build a command and control channel. In a large network environment, it is impractical for a botmaster to actively trace all of its bots. To evade detection, a botmaster normally does not actively contact or scan all bots, particularly when the botnet contains a large number of bots behind firewalls or NAT that would not freely allow incoming traffic.
- *Information dispersion/harvesting:* Sooner or later, a bot will be ordered to take some actions through the established command and control channel[1]. A bot can be asked to collect sensitive information from the local machine (*information harvesting*), or participate in organized attacks, such as spamming and DDoS attacks, against a third party (*information dispersion*)[2].

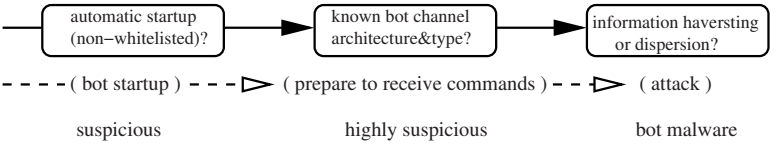


Fig. 1. BotTracer detection logic: startup, preparation, and attack during a bot onset

In this work, we propose BotTracer to detect bots by capturing these three invariant features during their execution. *First*, when a host starts, a virtual machine with the same image is also started. A virtual machine without any human interactions provides an effective playground to identify processes that are

[1] If a bot simply hibernates, it does not do any harm, although it makes bot detection more difficult.

[2] We do not consider information processing as defined in [19].

automatically started, especially those with networking activities. *Second*, with such a playground with little noise traffic at hand, BotTracer keeps monitoring automatic communications and classifies these communication channels. Since a bot must actively contact a rendezvous point to build a command and control channel with its controller, BotTracer capture these channels and compare them with known characteristics of bot command and control channels. This can significantly narrow down the detection space. *Lastly*, BotTracer constantly monitors system-level activities and traffic patterns of those processes that have been identified as suspicious. Hence, BotTracer is able to capture those bots that are actively performing information harvesting or dispersion. Figure 1 illustrates BotTracer's detection logic. To evaluate the performance of our proposed system, we implement a prototype of BotTracer based on VMware and Windows XP Professional and test a variety of bots. The experimental results demonstrate that BotTracer can detect all these bots without false negatives.

Comparisons with Existing Approaches: Most recently, BotHunter [20] is proposed to detect bots by correlating events from inbound scan to outbound scan. As BotHunter aims to detect bot behavior at the network level, stealthy bots can dodge detection by evading event timing correlation, or conducting local attacks (such as deleting files) without any networking activities. The work in [29] focuses on remote control behavior analysis by tracking tainted data from the network. Panorama [31] also relies on taint analysis to analyze malware behavior, although it is done after a suspicious sample has already been detected. A flip side of Panorama is that its effectiveness is contingent on the completeness of the samples that have been collected. SpyProxy [24] also employs behavior analysis through execution to detect malicious Web content. It is, however, based on evidence of malicious side-effects (the attack phase). Compared against these previous schemes, BotTracer leverages virtual machine technology to significantly reduce the detection space, and relies on in-depth bot behavior analysis to detect bots.

BotTracer takes a host-based approach, which complements existing network-based approaches. Efficiency of network-based approaches has already been challenged by various techniques, such as obfuscation [26] and encryption [12]. More importantly, a network-based approach commonly results in the shutdown of the command and control channel server or the change of the DNS entries [18]. Since it leaves infected machines unchanged, they can be easily reclaimed later. In our experiments, we have shown that for traditional centralized botnets, BotTracer is able to locate the centralized server after successfully identifying a bot machine.

The remainder of the paper is organized as follows. We present the design of BotTracer in Section 2. Based on the implemented prototype, we evaluate BotTracer in Section 3. We discuss some limitations in Section 4 and make concluding remarks in Section 5.

2 BotTracer Design

Although bots commonly exhibit the three fundamental features as we identified above, detecting these characteristics is challenging since bot-like malware

commonly employ various techniques to conceal themselves. Therefore, in BotTracer, a virtual machine that clones the host image is constructed to provide an ideal detection environment. That is, once the host is started, a virtual machine (VM) that clones the host image is also started automatically. However, the user only operates on the host. The virtual machine thus becomes an environment without human interactions.

On such a playground with significantly reduced noise, BotTracer thus focuses on detecting the three invariant bot behaviors through execution:

- Once the virtual machine in BotTracer starts, automatically started processes will self-expose. After filtering the processes on the whitelist, BotTracer keeps monitoring the remaining ones. As a bot must actively build a command and control channel to the outside before any malicious behavior is conducted, any outgoing traffic from any remaining process on the virtual machine indicates it is suspicious.
- By constantly monitoring its inbound and outbound traffic once a process is flagged as suspicious, BotTracer can identify whether a special command and control channel is established. For example, a traditional IRC-based bot may build a persistent connection to receive commands from the botmaster, while modern bots may periodically contact the botmaster.
- An identified command and control channel indicates a highly suspicious bot-like process. Since a bot will perform information harvesting or dispersion sooner or later, BotTracer, which constantly monitors the highly suspicious processes, detects information harvesting by tracing relevant system calls and the corresponding parameters that are intercepted through the virtual machine monitor, and detects information dispersion by inferring the traffic patterns.

The detection scheme described above is based on an *ideal* virtual machine environment. In practice, however, there are a number of issues that must be solved to make the detection effective.

2.1 Whitelist and Starting Point Set

In BotTracer, the absence of user interactions on the virtual machine dramatically reduces normal user traffic. To further facilitate the bot detection, it is preferable to eliminate the interference from some legitimate processes that are automatically started on the virtual machine as well. We classify them into three categories as [13]: system daemons, software updates, and network applications automatically started by the OS. The first category covers system daemons like `system.exe`, `svchost.exe`, and `services.exe` in Windows. The second refers to automatic software updates from the well-known Web sites. For the limited number of processes in the first and the second categories, as their networking behaviors are mostly fixed, we can simply whitelist them to allow their network connections.

The third includes user application processes like MSN and ICQ that are configured in advance. For example, if a MSN client is configured to sign in automatically, the MSN client will connect to the MSN server once it is started. In

addition, if a bot is bound to a popular application, such as a Web browser, the bot may not automatically start once the virtual machine starts, but starts when a particular application is started. For detection purpose, once they are started by a user on the host, they are started on the virtual machine as well by application synchronizer. For instance, a Web browser typically has a default setting that allows a user to visit a Web site once the Web browser is launched. Thus, once the Web browser is started, it will generate outbound traffic automatically. Since the number of such applications on a host is limited, it is also possible to have their default destinations, which we call *starting points*, whitelisted.

Although being effective most of the time, this approach may unnecessarily burden bot detection: by default a Web browser is set to connect to only one starting point, but the Web page of this site may contain rich information that leads to connections to other sites. If the Web page is frequently updated, this would be more difficult for us to whitelist the traffic based on one starting point. For other applications, the starting point might be a registry server that needs to validate user identification or a name service that provides references to other resources. Thus, allowing connections to the starting point may not only complicate whitelisting, but also make it inaccurate.

More importantly, there is a running copy of the application on the host. If the process on the virtual machine is allowed to send out traffic, it may affect the status of the application process on the host and lead to unexpected results. For example, the automatic sign-in to a service from the virtual machine process may kick out its corresponding host process that signed in before.

Therefore, to guarantee correct semantics of normal applications and ease bot detection, in addition to put them on the whitelist, we also block the connections of user applications in the third categories to their starting points once they are going through the virtual machine monitor. Blocking these connections can thwart a sequence of actions of the process, and can turn these processes into semi-dormancy or dormancy in most situations according to our experiments. Conservatives can even merge the whitelist into the starting point set. Thus, legitimate traffic and process activities could be minimized, which is very favorable for bot detection.

Generating whitelist and starting point set for system daemons and software updates is relatively easy. We found that nearly all of their network traffic is to well known destinations. BotTracer thus can collect traffic information on a typically configured Windows XP machine. Generating whitelist and the starting point set for network applications is more difficult. BotTracer needs to query starting points of popular applications, such as the default destination of a Web browser, with the intervention of users.

2.2 Command and Control Channel Detection

Now we present how BotTracer detects the initialization of a command and control channel in a controlled virtual machine with minimum noisy traffic. A bot always needs to actively build a command and control channel to communicate with its botmaster. In practice, there are two architectures for operating such channels.

1. **Centralized:** The first is a centralized architecture. Traditionally, IRC-based botnets commonly leverage IRC servers to issue commands to the army of bots. In this centralized mode, there are a number of varieties. For example, the destination could simply be a list of static IP addresses or a list of URLs so that flexible IP addresses could be used. Some bots, such as `Graybird` [6], may use an intermediate point, in which the bot will access a static URL, retrieve the actual centralized server address, and connect to it.
2. **Decentralized:** A more recently emerged architecture that has also been foreseen by many researchers for bot communications is through distributed networks, such as P2P. This decentralized architecture can reduce the risk of being detected. `Nugache` is such a Trojan that uses P2P technology for communication [28]. In addition, not all P2P bots need seed servers. For example, `Sinit` sends special discovery packets to look for peers [28].

Regardless of the architecture, there are two types of command and control channels:

1. **Type 1 – Persistent Channel:** In this approach, a bot process directly starts a connection to the destination and the connection is persistent. The average connection time could be as long as 3.5 hours according to [11]. This type of connections is normally initialized upon the startup of the machine. IRC bots commonly use this approach.
2. **Type 2 – Periodic/Sporadic Channel:** In this approach, the bot process periodically starts connections to a destination. Typically, the destination has not communicated with the host before. An easy variation of this type is to launch aperiodic connections instead of periodic ones. HTTP-based `Bobax` bot [8] falls into this category.

Given the command and control channel architectures and types, we can leverage these known characteristics to construct a bot channel event model. The bot channel initialization event model consists of two levels. The first level represents the channel type, indicated by low level events, such as a new connection is initialized, an incoming connection is accepted, and a connection is reset. The channel type level generates input to the channel architecture level, which represents whether a centralized channel is built, a decentralized channel is built, etc.

Some IRC-based botnets use persistent channels and the average duration of an IRC bot can be as long as 3.5 hours [11]. By contrast, a typical sporadic channel that uses HTTP may last for only a few seconds. Hence, we use the following heuristics to detect the channel type:

- A *new* connection refers to one whose destination has not been contacted before since the process has started.
- At the beginning, if a new connection is built, the connection is said to connect to an *intermediate* point.
- If an intermediate point is reconnected, the connection is updated to be a *sporadic* one.

– If a connection to an intermediate point or a sporadic connection lasts more than 30 seconds, it is flagged as a *persistent* one.
– When a new connection is accepted, it is flagged as a *sporadic* one.

Based on the above setup, Figure 2 shows the state transitions in our two-level model. Note that in the command and control channel detection, BotTracer focuses on detecting the establishment of a command and control channel without tracking how the channel is used. Therefore, it is expected that we would get some false positives, which BotTracer relies on the next step to further reduce.

2.3 Information Harvesting/Dispersion Behavior Analysis

Information Harvesting Detection. For information harvesting, a bot may be instructed to collect the information such as password, game/bank accounts, product keys, some personal information, and report to the botmaster. Some of such information may exist under particular application's directories. Some may be collected from the program's memory space when the application is running. Windows temporary files are also a popular target. For example, malware can search sensitive data in cookies. In addition, some system information is also attractive, such as registry entries. Existing research shows that currently information harvesting is mainly through code injection, keystroke log, and direct memory reading.

While designing strategies to detect and defeat each of these is possible, malware developers may invent new evading approaches. Instead, since information harvesting must involve disk or memory accesses, we rely on the process behavior analysis at the system level to detect information harvesting as follows.

Intuitively, if a bot is commanded to access the disk, no matter what approach it takes, monitoring disk accesses related system calls/APIs could identify any disk accesses. For Windows systems, BotTracer can monitor a limited number of critical system APIs, such as `OpenFile`, `CreateFileMapping`, `CreateFileMappingNuma`, and `OpenFileMapping`. Accessing any of these triggers an alarm.

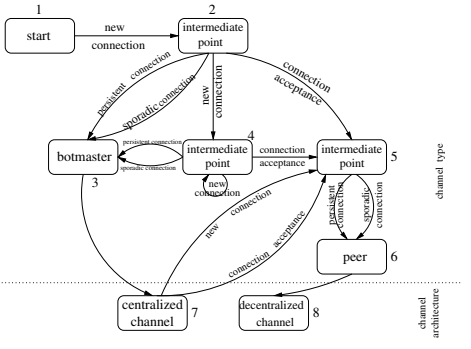


Fig. 2. Command and Control Channel Event Model

If a bot harvests information from a process' memory space, no matter which approach (code injection, keystroke log, or direct memory reading) is used, the malware typically starts from querying the information of the process or the window in order to locate the exact victim it wants to peek at. Thus BotTracer can monitor `OpenProcess`, `WriteProcessMemory`, `ReadProcessMemory`, `CreateRemoteThread`, `FindWindow`, `SetWindowsHookEx`, `GetWindowThread ProcessId`, `CreateToolHelp32Snapshot` and their family APIs for potential information harvesting.

Monitoring all these API calls for all processes on the virtual machine is cumbersome since there are a few whitelisted legitimate applications on the virtual machine. Among them, we are particularly concerned about the third category applications, because malware can inject their code into these popular applications. For these applications, they may have disk and memory activities, although their connections to their starting points are cut off (network accesses are not allowed). We have performed extensive experiments and the result shows that most of such processes do not have any further activities without network access, while a few do have disk and memory accesses occasionally. To deal with them, we define their profiles in advance. That is, for a limited number of automatically started processes or popular application processes that may be started by the application synchronizer, we generate their profiles after their connections to their starting points are forcefully cut off without user interactions or network accesses. We call such profiles as *dormant process profile* as most process activities are turned off. The profile includes the resources they can access, the system functions through which they access, etc. In detection, once a process behaves out of its profile, an alarm is raised. We extend the XML language to define the dormant process profile. Figure 3 in Appendix A shows the profile of the `Internet Explorer` that is generated on a clean machine without user interactions or network accesses.

Information Dispersion Detection. Besides information harvesting that endangers the infected machine, a bot is commonly commanded to participate in organized attacks, such as DDoS and spam. Many schemes have been proposed to deal with these attacks by leveraging some application level characteristics and are thus application-dependent. From the perspective of an attacking bot, however, all these attacks will show some unique traffic patterns. Furthermore, the traffic destination should not be in the starting point set or on the whitelist. Lastly, it is less likely such traffic is encrypted. Thus, we design our detection and thwarting scheme as follows.

A common feature of information dispersion attacks is that the target of the outgoing traffic is a third party, and often is a destination that the bot has not communicated with before. As BotTracer starts to monitor the process behavior from the beginning, it has the record of all the destinations that the bot has communicated with. Before the attack is launched, the communication through the command and control channel is bi-directional. After the bot receives the attack command from the botmaster, the outgoing traffic is likely to go to new destinations. Thus, at a higher level without interpreting any communication

Table 1. Command and Control Channel Detection

Name	Alarm Time (s)	Architecture	Type
Agobot	6.532 seconds	Centralized	Persistent
Forbot	34.173 seconds	Centralized	Persistent
Jrbot	1.895 seconds	Centralized	Persistent
Reptilebot	2.719 seconds	Centralized	Persistent
Sdbot	0.953 seconds	Centralized	Persistent
Rxbot	4.409 seconds	Centralized	Persistent
Graybird	2.997 seconds	Centralized	Persistent
Nugache	1.422 seconds	Suspicious	Suspicious

content, the destination of the outgoing traffic is different from the previous incoming one, and an asymmetric traffic pattern could be observed.

However, only monitoring outgoing traffic patterns is not sufficient. Recall that on the virtual machine, there is no user interaction and most of the legitimate applications are semi-dormant or dormant. Thus, if there is outgoing traffic from a process on the virtual machine, we can further examine its profile. If the destination is not a starting point specified in the profile, it is highly likely that the process is hijacked by a malware. Moreover, as there is no human interactions on the virtual machine, if there is outgoing email traffic, very likely it is generated due to spam attacks. Lastly, although the bot normally does not generate a large amount of traffic, once it participates in a DDoS attack, its traffic amount would increase remarkably in a short period, which can be leveraged to detect DDoS attacks.

3 BotTracer Evaluation

Based on our design, we have implemented a prototype of BotTracer. In this section, we present the experimental results of BotTracer when a set of representative bots are tested, including the following three classes of bots:

- `IRC bots and their variants` are traditional bots controlled through IRC. We tested a variety of IRC bots including `Agobot4 private`, `Forbot`, `Jrbot`, `Sdbot`, `Reptilebot`, and `Rxbot`.
- `Graybird` has a large number of variants since its first debut in 2001. It is one of the most prolific pieces of Windows malware. We experimented on version 2005. It does not use IRC, but its own communication protocol. To hide itself, it injects itself to `Internet Explorer` (IE). Our testing version can start an IE process and copy itself to IE space and then execute in the context of IE.
- `Nugache` uses encrypted and/or obfuscated P2P traffic for communication. It opens TCP port 8 and has a static list of 22 initial peers to which a peer aims to connect to at TCP port 8. After successful connection, it is going to exchange the list of successfully connected peers. It participates in DDoS

attacks once commanded, and it spreads over instant messengers such as American Online Instant Messenger [28]. In Windows, it runs as a `mstc.exe` after infection.

In addition, `Microsoft Outlook Express` and `pcAnywhere` are also experimented to study false positives.

3.1 Prototype Implementation and Experimental Setup

We have implemented the prototype based on Windows XP Professional. Particularly, we use VMware workstation version 5.5.3 for the virtual machine. We use VMware `Converter` [3] to clone the physical machine. The traffic pattern monitor and analyzer are implemented in a *traffic* module, and the process behavior analyzer is implemented as a separate *behavior* module.

The traffic module monitors all ingress and egress traffic after an application starts. As it is necessary to map ports to the owning process for further analysis, we implement our traffic module based on `Enhance Netstat` [4]. It can map a port to its owing process even if the process adopts some approaches to hide itself from `Task Manager`. When the channel architecture and type cannot be detected and there is outgoing traffic that is not going to a starting point, BotTracer reports it as suspicious.

The behavior module is implemented based on Microsoft `Detours 2.1 Express` [1]. It intercepts Win32 function calls. For our experiments, the behavior module is designed to capture all violations of the sensitive data accesses that are not allowed in the dormant process profile. In our current implementation, it monitors a limited number of Win32 functions that are for file and network accesses. In addition, process management functions are monitored.

For experiments, we have set up a controlled network. BotTracer was run on a machine with a 2.79 GHz CPU and 2 GB RAM. The guest OS of VMware is Windows XP Professional that is identical to the host OS.

`Graybird` injects itself to IE at runtime and its dormant profile is shown in Figure 3. For `IRC based bots`, we set up an IRC server on another machine with similar configurations and we modified source code to direct bot samples to our IRC server so that we can issue commands to the bot through a connected IRC client. `Graybird` is configured with its GUI tool. Its botmaster runs on another machine. For `Nugache`, because only binary is available, we can do few configurations.

3.2 Channel Establishment Detection

In the controlled environment, we first test whether BotTracer can successfully detect the channel establishment and the corresponding channel type and architecture. Table 1 shows the detection results for the eight bots. *Alarm Time* is the time between when the bot starts and when its first outgoing traffic is captured.

Table 2. APIs called when `Rxbot` launches attacks

Action	API	Arguments
Access Registry	RegOpenKeyEx RegQueryValueEx	Software\BioWare\NWN\Neverwinter Location
Access Directory	fopen fget	C:\NeverwinterNights\NWN\nwncdkey.ini file handle

We found that nearly all bots initialized the command and control channel within 10 seconds after their startup. Furthermore, both `IRC bots` and `Graybird` establish one and only one persistent TCP connection. The entire channel detection time is less than 60 seconds. Note for `Nugache`, as the bot tries to connect to 22 initial peers that are hardcoded in the binary, all these connections failed as expected. BotTracer thus cannot report the architecture and type of the channel. However, since it tries three times for a destination and tries different new destinations in a sequence, BotTracer still reports it as suspicious.

Furthermore, as centralized channels are detected for `IRC bots` and `Graybird`, we check whether or not the host (not the virtual machine) has connections to the same IP and port because on the host there are also identical bot copies. We found both `IRC bots` and `Graybird` on the host also connect to our IRC server and `Graybird` botmaster, respectively. This confirms that a running copy of the bot process on the virtual machine does not affect its corresponding process running on the host. Furthermore, for bots operated in the centralized mode, it is straightforward to further trace down to the server and shut down the server, and possibly the entire botnet.

3.3 Information Harvesting/Dispersion Detection

As BotTracer alarms for all eight bots, the behavior module is activated as well (note the traffic module is still active in order to thwart potential attacks). Unlike the channel detection which is completed in a short time after a process starts, a bot usually performs information harvesting or dispersion only after it receives commands from the botmaster. Thus, through our experimental setup, we act as the botmaster to start attacks. Particularly, for information harvesting attacks, `Rxbot` was instructed to return keys of products, via a `getcdkeys` command. For information dispersion, we launch an information dispersion attack through `Agobot` by sending it a DDoS command. For `Nugache`, we failed to send any command as we do not have the source code and its behavior is not well understood. In any of the experiments, we keep the logs of traffic and system activities.

Table 2 shows the intercepted APIs and the corresponding parameters for the `Rxbot` process after `getcdkeys` is received.

Since `Rxbot` has already been reported to be highly suspicious, accessing registry and files under an application directory in the above actions leads BotTracer to report it as a bot and disable its input and output.

Table 3. Agobot HTTP DDoS Attack Packets

Time (s)	Source	Destination	Type
0	192.168.88.156	192.168.88.155	IRC
0.012	192.168.88.155	192.168.93.52	HTTP
2.608	192.168.88.155	192.168.93.52	HTTP
5.226	192.168.88.155	192.168.93.52	HTTP

Table 3 shows the packet sequences once a `.ddos.httpflood http://www.aaaaa.com 100 www.aaa.com 2000`[3] command is issued to `Agobot`, which requests the `Agobot` to send 100 HTTP requests to `www.aaaaa.com` with a 2000 ms interval. `www.aaa.com` is the HTTP referer. Note that `192.168.88.156` is IP of the IRC server. The IP of the virtual machine is `192.168.88.155`. The attack target uses `192.168.93.52`.

In the experiment, `Agobot` sends out each HTTP attack packet for 100 times. Table 3 gives the time BotTracer takes to capture the outgoing attack traffic. In our implementation, both the traffic pattern monitoring and the starting point (to compare the outgoing traffic destination) in the process profile are leveraged. The default threshold of outgoing packet number is 3, which means it takes 5.2 seconds for BotTracer to detect the attack. Conservative protection can reduce the threshold to 1.

3.4 False Positive Experiments

False positives occur when normal applications are flagged as bot malware. Maintaining and timely updating the dormant profile list for the normal and popular applications on a host can greatly reduce false positive. We first test whether or not a normal application without a profile can be captured in BotTracer. On the host running BotTracer, we install `Microsoft Outlook Express 6`. We set it up to check a `hotmail` mailbox once every minute, and the account and the password are saved before the experiment was run. BotTracer quickly reports this is a bot using a centralized and sporadic channel!

We disable BotTracer, and run it again. We have the following first six packet sequence log as shown in Table 4 without any user interactions. In this table, the first packet is a DNS query for `services.msn.com`. `192.168.68.227` is a domain name server. The application thus obtains the IP address `64.4.60.7`. `Outlook Express` does not keep a persistent TCP connection. Instead, about every one minute it starts a new TCP connection to the Web email server `65.55.154.125` and checks for new emails. This pattern causes BotTracer to report a false positive.

To validate whether a user could add its profile to BotTracer to eliminate false alarms, we generate the dormant process profile for `Outlook Express`, and `Outlook Express` is started again in BotTracer. As expected, BotTracer did

[3] URL and the public IP addresses are anonymized. The prefix of the public IP address is replaced with 192.168 when necessary.

Table 4. Outlook Express 6 Connecting Packets

Time (s)	Source	Destination	Type
0	192.168.88.155	192.168.68.227	DNS
0.095	192.168.88.155	64.4.60.7	HTTP
0.478	192.168.88.155	65.55.154.125	HTTP
1.129	192.168.88.155	65.54.183.193	HTTP
63.908	192.168.88.155	65.55.154.125	HTTP
124.463	192.168.88.155	65.55.154.125	HTTP

not raise an alarm. These indicate that it is critical for the user of BotTracer to update the profile list once new applications are installed.

In addition to `Microsoft Outlook Express 6`, `pcAnywhere 12.0.0` is run to see if false positives would be raised. Controlled by a `pcAnywehere` remote, `pcAnywhere` host has similar functions as a `Graybird` bot. Both a `pcAnywehere` remote and a `Graybird` botmaster can manipulate nearly all computer resources under control. We ran both of them on BotTracer. As before, BotTracer successfully detects the command and control channel of `Graybird`, which is flagged as centralized and consistent, while no alarm is raised for `pcAnywhere`. The critical reason for this result is that a `pcAnywhere` remote requires the contact information of a `pcAnywhere` host, while a `Graybird` bot requires the contact information of the `Graybird` botmaster. That is, a `pcAnywhere` host waits to be connected by a `pcAnywhere` remote while a `Graybird` botmaster waits to be connected by `Graybird` bots.

These case studies just show that it is possible to reduce false positives through accumulated process profiles. However, in practice, we believe false positives and false negatives would be inevitable, particularly when new techniques are continuously adopted by bot developers.

4 BotTracer Limitations

A fundamental assumption of BotTracer is that the virtual machine cannot be detected by the bot. In practice, there are many techniques that can detect virtual machines [7] [32]. Thus, if a bot detects whether it is running on a virtual machine based system, our BotTracer will not work properly. This issue can be addressed from two perspectives. *First*, as this is a challenge for all virtual machine based solutions, anti-fingerprinting techniques are still improving [5]. For BotTracer, the detection behavior of a bot could be detected through system level activity monitoring, and thus provides an opportunity to cheat the bot. *Second*, the adoption of virtual machines in practice is quickly increasing [2]. With the continuous performance improvement of the virtual machines, such as VMware and Xen, and the pervasive availability of dual core processors, running applications in virtual machines may slightly degrade the user performance. Therefore, in BotTracer, we can run applications in virtual machines instead. *Lastly*, most bots currently do not detect the virtual machine based honeypots [30].

On the other hand, we prohibit user behavior on the virtual machine in order to make automatically started process self-exposed. If a bot first detects user activities before it launches itself, the current BotTracer would fail to detect such bots. The countermeasure is to synchronize user actions on the host with the corresponding applications on the virtual machine [10].

As always, developing and thwarting bot-like malware is endless arm race. It is foreseeable that some bots with new techniques may evade the detection of BotTracer. For example, if the bot developers use a scheme to identify all bots by labeling each bot with a unique ID when the bot first registers, the botmaster is able to detect the simultaneous arrivals of two bots with the same ID if BotTracer is activated. It is also difficult for BotTracer to detect time-bomb bots. Moreover, as BotTracer relies on known characteristics of bot malware, bots equipped with alternative approaches [23] (e.g., the communication channel is started by user operations when a malicious Web page is accessed) can evade its detection. We are currently trying improve BotTracer for better detection accuracy.

5 Conclusion

Bots and botnets have attracted a lot of attention from both the industry and research communities recently. Detecting bots, however, is still very challenging since bots are very stealthy and bot developers continuously and quickly adopt new techniques to evade detection. In this study, we propose BotTracer to effectively detect bot-like malware on end systems through detecting the bot startup, preparation, and attack behavior during execution. A prototype of BotTracer has been implemented based on VMware and a set of representative bots are tested. The experimental results show that BotTracer is effective for bot detection.

Acknowledgements

We thank the anonymous referees for providing constructive comments. The work has been supported in part by the U. S. National Science Foundation under grants CNS-0509061, CNS-0621631, and CNS-0746649.

References

1. http://research.microsoft.com/sn/detours/
2. http://www.technologynewsdaily.com/node/4859
3. Convert physical machines to virtual machines, http://www.vmware.com/products/converter/
4. Enhance netstat - the code project, http://www.codeproject.com/internet/enetstatasp.asp
5. Malware immunization through deterrence and diversion, http://www.nsf.gov/awardsearch/showAward.do?AwardNumber=0650386
6. One of the most prolific pieces of windows malware has expired, http://news.softpedia.com/news/One-of-the-Most-Prolific-Piece-of-Windows-Malware-Has-Expired-51466.shtml

7. Honeyd security advisory 2004-001: Remonte detection via simple probe packet (2004), http://www.honeyd.org/adv.2004-01.asc
8. Taxonomy of botnet threats (November 2006), http://us.trendmicro.com/imperia/md/content/us/pdf/threats/securitylibrary/botnettaxonomywhitepapernovember2006.pdf
9. Barford, P., Yagneswaran, V.: An inside look at botnets (2006)
10. Borders, K., Zhao, X., Prakash, A.: Siren: Catching evasive malware. In: Proceedings of the IEEE Symposium on Security and Privacy, Berkeley, CA (November 2006)
11. Chen, Y.: High-performance network anomaly/intrusion detection and mitigation system (hpnaidm). In: ARO-DARPA-DHS Special Workshop on Botnets, Arlington, VA (June 2006)
12. Chiang, K., Lloyd, L.: A case study of the rustock rootkit and spam bot. In: Proceedings of the First Workshop on Hot Topics in Understanding Botnets, Cambridge, MA (April 2007)
13. Cui, W., Katz, R.H., Tan, W.: Binder: An extrusion-based break-in detector for personal computers. In: Proceedings of USENIX (2005)
14. Dagon, D.: The network is the infection (2005), http://www.caida.org/projects/oarc/200507/slides/oarc0507-Dagon.pdf
15. Dagon, D., Zhou, C., Lee, W.: Modeling botnet propagation using time zones. In: Proceedings of The 13th Annual Network and Distributed System Security Symposium, San Diego, CA (Febuary 2006)
16. Daswani, N., Stoppelman, M.: The Google Click Quality, and Security Teams. The anatomy of clickbot.a. In: Proceedings of the First Workshop on Hot Topics in Understanding Botnets, Cambridge, MA (April 2007)
17. Freiling, F., Holz, T., Wicherski, G.: Botnet tracking: Exploring a root-cause methodology to prevent distributed denial-of-service attacks. In: Proceedings of the 10th European Symposium on Research in Computer Security (ESORICS) (September 2005)
18. Goebel, J., Holz, T.: Rishi: Identify bot contaminated hosts by irc nickname evaluation. In: Proceedings of the First Workshop on Hot Topics in Understanding Botnets, Cambridge, MA (April 2007)
19. Grizzard, J., Sharma, V., Nunnery, C., Kang, B., Dagon, D.: Peer-to-peer botnets: Overview and case study. In: Proceedings of the First Workshop on Hot Topics in Understanding Botnets, Cambridge, MA (April 2007)
20. Gu, G., Porras, P., Yegneswaran, V., Fong, M., Lee, W.: Bothunter: Detecting malware infection through ids-driven dialog correlation. In: Proceedings of 16th USENIX Security Symposium, Santa Clara, CA (June 2007)
21. Karasaridis, A., Rexroad, B., Hoeflin, D.: Wide-scale botnet detection and characterization. In: Proceedings of the First Workshop on Hot Topics in Understanding Botnets, Cambridge, MA (April 2007)
22. Kawamoto, D.: Bots slim down to get tough. CNET News.com (November 2005)
23. Lam, V.T., Antonatos, S., Akritidis, P., Anagnostakis, K.G.: Puppetnets: Misusing web browsers as a distributed attack infrastructure. In: Proceedings of ACM CCS (2006)
24. Moshchuk, A., Bragin, T., Deville, D., Gribble, S., Levy, H.: Spyproxy: Execution-based detection of malicious web content. In: Proceedings of the 16th USENIX Security Symposium, Boston, MA (August 2007)
25. The Honeynet Project. Know your enemy: Tracking botnets (March 2005), http://www.honeynet.org/papers/bots

26. Provos, N., McNamee, D., Mavrommatis, P., Wang, K., Modadugu, N.: The ghost in the browser analysis of web-based malware. In: Proceedings of the First Workshop on Hot Topics in Understanding Botnets, Cambridge, MA (April 2007)
27. Rajab, M., Zarfoss, J., Monrose, F., Terzis, A.: A multifaceted approach to understanding the botnet phenomenon. In: Proceedings of Internet Measurement Conference (IMC), Rio de Janeiro, Brazil (October 2006)
28. Schoof, R., Koning, R.: Detecting peer-to-peer botnets (Feburary 2007), http://staff.science.uva.nl/~delaat/sne-2006-2007/p17/report.pdf
29. Stinson, E., Mitchell, J.C.: Characterizing the remote control behavior of bots. In: Hämmerli, B.M., Sommer, R. (eds.) DIMVA 2007. LNCS, vol. 4579. Springer, Heidelberg (2007)
30. Wang, P., Sparks, S., Zou, C.: An advanced hybrid peer-to-peer botnet. In: Proceedings of the First Workshop on Hot Topics in Understanding Botnets, Cambridge, MA (April 2007)
31. Yin, H., Song, D., Egele, M., Kruegel, C., Kirda, E.: Panorama: Capturing system-wide information flow for malware detection and analysis. In: Proceedings of the 14th ACM Conference on Computer and Communication Security, Alexandria, VA (October 2007)
32. Zou, C., Cunningham, R.: Honeybot-aware advanced botnet construction and maintenance. In: Proceedings of the International Conference on Dependable Systems and Networks (DSN) (June 2006)

Appendix A

Figure 3 depicts the profile of the `Internet Explorer` that is generated on a clean machine without user interactions or network accesses.

```
<profile>
    <name>
        Internet Explorer
    </name>
    <description>
        the profile of Microsoft Internet Explorer
    </description>
    <path>
        C:\Program Files\Internet Explorer\iexplore.exe
    </path>
    <starting point>
        www.google.com
    </starting point>
    <registry>
        no
    </registry>
    <file access function>
        getFileSize
    </file access function>
    <file access path>
        C:\Documents and Settings\user\Local Settings
        \Temporary Internet Files\Content.IE5\index.dat
    </file access path>
    <alarm>
        yes
    </alarm>
</profile>
```

Fig. 3. The dormant profile of `Internet Explorer`

Towards Automatically Generating Double-Free Vulnerability Signatures Using Petri Nets

Ryan Iwahashi[1], Daniela A.S. de Oliveira[1], S. Felix Wu[1],
Jedidiah R. Crandall[2], Young-Jun Heo[3], Jin-Tae Oh[3], and Jong-Soo Jang[3]

[1] University of California at Davis
[2] University of New Mexico
[3] ETRI

Abstract. With the increased popularity of polymorphic and register spring attacks, exploit signatures intrusion detection systems (IDS) can no longer rely only on exploit signatures. Vulnerability signatures that pattern match based on properties of the vulnerability instead of the exploit should be employed. Recent research has proposed three classes of vulnerability signatures but its approach cannot address complex vulnerabilities such as the ASN.1 Double-Free. Here we introduce Petri nets as a new class of vulnerability signature that could potentially be used to detect other types of vulnerabilities. Petri nets can be automatically generated and are represented as a graph making it easier to understand and debug. We analyzed it along side the three other classes of vulnerability signatures in relation to the Windows ASN.1 vulnerability. The results were very promising due to the very low false positive rate and 0% false negative rate. We have shown that Petri nets are a very efficient, concise, and effective way of describing signatures (both vulnerability and exploit). They are more powerful than regular expressions and still efficient enough to be practical. Comparing with the other classes, only Turing machines provided a better identification rate but they incur significant performance overhead.

1 Introduction

With the increased popularity of polymorphic and register spring attacks, intrusion detection systems (IDS) can no longer rely only on exploit signatures. Vulnerability signatures that pattern match based on properties of the vulnerability instead of the exploit should be employed. Third generation vulnerabilities [17], such as double free, still are unpatched, for the most part. Although there a great number of works in the literature focusing on traditional classes of vulnerabilities, such as buffer overflows and format strings, double-frees have not received much attention.

Host-based intrusion detection systems (HIDS) have been shown successful in stopping many worm attacks [4, 5, 9, 13, 15, 19] but there is still a desire to detect potential attacks at the network level. Network-based intrusion detection systems (NIDS) have the potential of reducing network traffic and increasing host efficiency and rely heavily on signatures.

T.-C. Wu et al. (Eds.): ISC 2008, LNCS 5222, pp. 114–130, 2008.

Exploit-based signatures based on pattern-matching have been show of limited effectiveness in detecting worm attacks. The problem is that many different attacks can exploit the same vulnerability. In addition, polymorphism and metamorphism exponentially expand the required number of signatures for a single exploit [6]. Consequently, recent research has started to focus on signatures based on the vulnerability [3, 6]. However, recent approach [3] cannot be applied to complex vulnerabilities such as ASN.1 Double-Free.

In this paper we propose a new class of signature using Petri nets that addresses double-free vulnerabilities such as the ASN.1 vulnerability. It can potentially be applied to other types of vulnerabilities such as buffer overflows and format strings and can be automatically generated using an intrusion detection system and a symbolic execution tool.

The rest of the paper is organized as follows. Section 2 discusses vulnerability and exploits signatures. Section 3 describes the ASN.1 Double-Free vulnerability. Section 4 discusses in detail our Petri net approach. In Section 5 we present the tests we have performed to evaluate the effectiveness of our approach using the ASN.1 vulnerability. We have also compared our approach, in relation to ASN.1, with other three classes of vulnerability signatures presented in the literature. Section 6 discusses related work. Our conclusions and future work are presented in Section 7.

2 Vulnerability Signatures Versus Exploit Signatures

This paper deals with vulnerability signatures as opposed to exploit signatures, and here, we explain the difference between a vulnerability and an exploit; these terms are often used interchangeably although they are completely different.

Vulnerabilities are errors, flaws, weaknesses and lapses of security introduced to computer systems during the software engineering process (requirements, analysis, design, implementation, tests, operation and maintenance) that allow unauthorized access and actions into the system (vulnerability exploitation) [2]. Some of the worst vulnerabilities allow attackers (those who explore vulnerabilities) to run their code on the compromised system. A vulnerability signature must not be based on malicious code or return addresses because these aspects can be different for every exploit or altered using polymorphism or register springs (use of known, static jumps in the source code to jump to malicious code), for instance [6, 12]. This means that a vulnerability signature must be based on properties of the vulnerability instead of the properties of the exploit. For example, a buffer overflow will always need to send too many bytes into a limited buffer to overwrite data on the stack. While exploit signatures will look for data that is in the buffer, a vulnerability signature will look for any string of bytes that is long enough to overrun the buffer.

An exploit is an instance of code or bytes conforming to a certain application protocol that explores a vulnerability in a networked application so that management information (stack pointers, heap management pointers, user identity data, configuration data and so on) is corrupted with malicious data. When this data

is used by the architecture, the OS or the vulnerable program the malware is activated and allows an attacker to take control of the system. Exploit packets are those that triggered the execution of the exploit and, in practice, they are used as exploit signatures. Signatures for exploits have become the most common way of blocking malicious attacks primarily due to the fact that they are the easiest to derive out of the exploit traces. They can be created by searching for patterns within the exploit packets based on the attack. The problem is that many different attacks can exploit the same vulnerability. In addition, polymorphism and metamorphism exponentially expand the required number of signatures for a single exploit [6]. A polymorphic exploit usually alters unimportant bytes or use encryption, whereas a metamorphic exploit uses equivalent instructions to achieve the same result. Despite the great number of possible exploits, the number of vulnerabilities is always finite. Therefore, creating signatures based on vulnerabilities instead of exploits is far more effective.

In spite of that, there is still some ambiguity left based on the fact that the exact vulnerability may be unknown without knowing the intention of the code programmers. For example, the ASN.1 double free vulnerability is caused because a freed chunk is returned from realloc when it is expected to be allocated. The vulnerability could either be the problem with realloc, because it should never return a freed chunk, or a problem with the ASN.1 function because realloc was used incorrectly. As a result, a vulnerability signature will not necessarily detect all exploits for a certain vulnerability, but all exploits for a vulnerable piece of code. Here we define *vulnerability signature* as a signature based on the properties of a vulnerable piece of code that allow any exploit targeting that piece of code to be successful.

Vulnerability signatures should have three key properties in order to be considered a good signature. First, the signature should not allow any false negatives. This is crucial because if a single attack is not detected by the signature then the entire network could be compromised. Second, the signature should match very few false positives because false positives will interfere with typical operations of the network. Too many false positives can act like a denial of service attack. Third, the signature matching time should not create any significant overhead. In this paper, these three properties will be used to evaluate our new vulnerability class and also to compare our approach with three vulnerability classes proposed in the literature [3]. Additionally, the signature should be automatically generated. The primary reason for this is that it may be infeasible to manually create a signature in a reasonable amount of time. Automatic generation of the signatures will ensure that networks are protected as quickly as possible. Also, the signature should be easy to manually understand in case analysis or modification is required.

3 The ASN.1 Double-Free Vulnerability

The ASN.1 Windows Vulnerability (MS04-007) is a double-free vulnerability announced in February 2004 and is known to affect almost all unpatched Windows operating systems including: Microsoft Windows 98, 2000, ME, XP, and 2003

[22]. The bug has the potential to allow a malicious hacker to obtain administrative access to the machine. In addition the vulnerability properties of the attack are completely data and control flow sensitive and can easily be made polymorphic.

Abstract Syntax Notation 1 (ASN.1) [10] is used to ensure that a message is interpreted by the receiver in the exact way the sender intended it to be. The syntax is a well defined set of rules and is utilized by many widely used Windows services including Microsoft's Internet Information Services (IIS), Exchange SMTP server, and Server Message Block (SMB). The basic encoding rules (BER) of ASN.1 build the message using various well defined types. The basic format is **type(1 byte) [Parameters of type] Data**. For example, a bit string type has the tag 0x03 and takes in two parameters of one byte each. The first parameter is the length (in bytes) and the second parameter is the number of unused bytes in the string.

3.1 Vulnerability Description

The ASN.1 vulnerability occurs while decoding certain ASN.1 messages that utilize the constructed bitstrings type. A constructed bitstring has the tag 0x23 and takes in a single one byte length parameter. The size of a constructed bitstring can be expanded the same way the size of a bitstring is expanded. Within the constructed bitstring there can be two subtypes: a bitstring or another constructed bitstring.

Whenever Windows receives an ASN.1 message it will use functions in MSASN1.dll to decode the message. The function that decodes constructed bitstrings contains a double free vulnerability [22]. A double free attack occurs when a chunk that is already freed gets freed again. For example, a programmer first allocates 8 bytes of memory for the variable temp which gets stored in chunk C. After using the space, the programmer later frees temp. After it is freed, temp will be inserted into the free list (a doubly linked data structure of freed chunks of memory). The forward and backward pointers of the free list are stored in the first 8 data bytes of the freed chunk (Figure 1). After it has been freed, the programmer

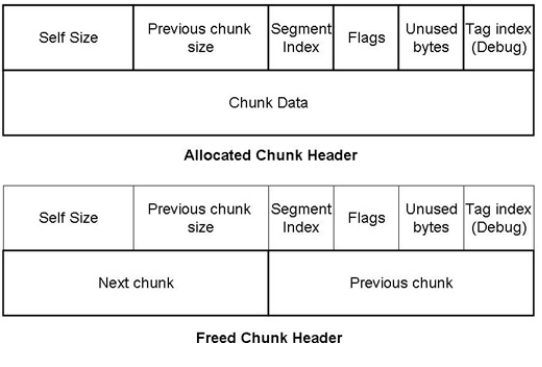


Fig. 1. Chunk Headers in a Windows Heap

mistakenly uses temp and writes X in the first 4 bytes and Y in the second 4 bytes of temp. Then the programmer frees temp again. The system will then try to insert temp into the free list for a second time right before where it is already inserted. The resulting link and unlink commands that are typical for a doubly linked data structure will then execute. However, since the forward and backward pointers have been overwritten it will produce different results. If you are trying to insert chunk A before B the link and unlink will take 4 operations. First, A->fw gets set to B. Second, A->bk gets set to B->bk. Third, B->bk->fw gets set to A. Last, B->bk gets set to A. In the corrupted case where chunk C is inserted before chunk C, first C->fw gets set to C. This will just make C->fw point to itself. The second step will cause C->bk to be set to C->bk. Third, C->bk->fw will be set to C. But C->bk has been set to Y. So Y->fw, or the first 4 bytes in Y will be set to the address of C. This is the key step because if Y is the address of a function pointer it will now call C instead of whatever function it is supposed to call. Then lastly, C->bk will be set to C. This means that C->fw and C->bk will both end up pointing to itself (Figure 2). If the data stored in memory location Y is a function pointer, the control flow will be altered when the function pointer is called.

Since double free vulnerabilities are well known, most commercial programs do not permit arbitrary write to freed chunks on the heap. However, due to a property of the realloc function the ASN.1 decoder in Windows allows a write to a pointer that is incorrectly assumed to point to allocated space. When decoding constructed bitstrings in the Windows ASN.1 decoder, it will be necessary to allocate space for any bitstrings present based on the given length. The problem will occur in only certain circumstances when the ASN.1 bitstring decoder is called; it will be called whether the bitstring is constructed or not. If it is constructed it will be called recursively to process embedded bitstrings. The bug is present within a constructed bitstring after at least one bitstring has been processed. Then if that bitstring is followed by a constructed bitstring and a bitstring inside the error will occur. This is because after any bitstring is processed then the bitstring buffer will not be freed. However, after a constructed bitstring is detected the bitstring buffer will be freed. This will cause realloc to return the freed pointer and allow overwriting of the free list pointers.

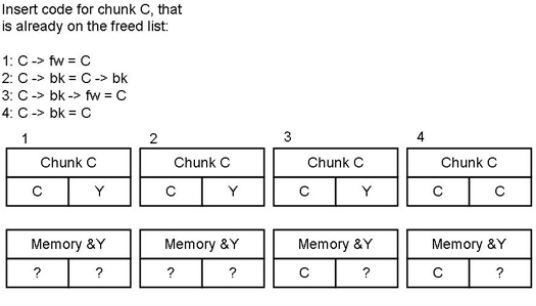


Fig. 2. Write to Arbitrary Memory Location Using a Double Free Attack

There are many aspects that make Windows ASN.1 attack particularly interesting. First, it is an example of a double-free vulnerability which was discovered much later than buffer overflows and format string vulnerabilities and are still present in many programs. They are much harder to detect statically and to perform signature generation than buffer overflow and format string vulnerabilities. Further, this vulnerability is control flow sensitive, data flow sensitive, and easily made polymorphic or metamorphic, which makes creating a strong signature for it particularly difficult. We explain these aspects below.

The control flow of the program refers to the order in which the individual instructions are executed. Control statements such as branches and conditional jumps can alter the control flow of a program. An attack is control flow sensitive if the success of the attack is affected by which branches are taken or not taken. In this case, altering a single bit can be the difference between a valid packet and an attack packet because the control flow will be altered. A control flow sensitive attack means that without a very strong signature that takes into account strict ordering then a large number of false positives will be obtained. Other vulnerabilities such as buffer overflows and format strings may not have this property. For example, most buffer overflows will not be affected by changing the control flow. If the same number of bytes is in the packet then the buffer will still be overrun. The same can be said about format string attacks. The data flow of a program refers to how the data of the packet is interpreted. The ASN.1 vulnerability is data flow sensitive because the value of one byte of data effects how other bytes of the data are interpreted.

Recent work [6, 12] has already shown that exploits can be very easily made polymorphic through the use of encryption and register springs. The ASN.1 vulnerability can also be made polymorphic as well. For example, the constructed bitstring that is used in the Kill-Bill exploit [7] can be made polymorphic by randomizing the bits and encrypting the exploit code. Also the invalid address and the overwritten function pointers can be varied. Although the vulnerability requires the use of constructed bitstring and bitstring tags, there are three properties that can be easily changed to make the vulnerability more complicated. First, the bitstrings can be of any size. Additionally, the use of the unused bytes tag can make it even easier to vary the sizes without ever changing the size of the exploit string. Second, there can be one or more constructed bitstring tags in between the two bitstrings. Third, the bitstrings that cause realloc to return a freed pointer can be put anywhere into the constructed bitstring. So the exploit bitstrings can be placed at the very beginning or embedded within multiple constructed bitstrings. These properties of the vulnerability create additional complications in creating a powerful signature.

4 A Petri Net Approach

A Petri net is a graphical and mathematical modeling tool and language used to represent discrete distributed systems[11]. As a modeling language, it graphically depicts the structure of a distributed system as a directed bipartite graph with

annotations and has place nodes, transition nodes, and directed arcs connecting places with transitions [21]. In this instance they will be used to generate a signature for the ASN.1 vulnerability. The key advantage here will be the ability of Petri nets to keep state and thus the important sizes and loops can be tracked. Although Petri nets are complex, even complicated vulnerability signatures, like the ASN.1 signature, can be represented in a reasonable size graph.

A petri net graph is basically comprised of five things: places, transitions, arcs, weights, and tokens. Tokens are used to help keep state, and there can be 0 or more tokens at any place. The places, represented by circles, are basically the different states of the graph. Transitions, represented by rectangles, are the processes that occur in order for tokens to change places. The arcs are the arrows that represent which places are the inputs for a transition and which are the outputs. The weights indicate how many tokens from each place are required to allow the transition to trigger. Figure 3 has a small example of matching strings that are of the form a^N b^N. Graphs are a very convenient way to express Petri nets, but more precision is required to accurately create good signatures. Formally a Petri net can be represented by a 5-tuple, PN = P, T, F, W, M_0 where: P = { p_1, p_2, ..., p_m} is a finite set of places, T = { t_1, t_2, ..., p_m} is a finite set of transitions, F $\subseteq$ (P $\times$ T) $\cup$ (T $\times$ P) is a set of arcs (flow relation), W: F $\rightarrow$ { 1, 2, 3, ...} is a weight function and M_0: P $\rightarrow$ { 0, 1, 2, 3, ...} is the initial marking.

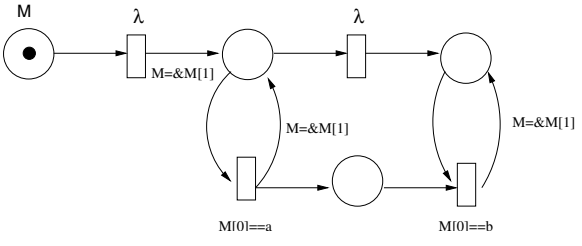


Fig. 3. Petri Net Example: Matches Strings of the form a^N b^N

4.1 Symbolic Execution

We have extended the DACODA symbolic execution tool [6] to automatically generate the Petri net signature for the ASN.1 vulnerability. DACODA is an extension of the Bochs virtual machine and can track down network bytes by performing full-system symbolic execution [8] on every machine instruction. It runs integrated with the Minos intrusion detection system [5] and can analyze an attack at the moment it is detected. Minos is a microarchitecture that implements Biba's low-water-mark integrity policy [1] on individual words of data and can catch zero-day control-flow hijacking attacks. In this microarchitecture, every 32-bit word of memory and general-purpose register is augmented with an integrity bit, which is set by the kernel when it writes data into them. This bit is set to low or high, depending on the trust the kernel has for it. Data coming from

the network are usually regarded as low integrity. Any control transfer involving untrusted data is considered a vulnerability, and a hardware exception traps to the VM whenever this occurs. DACODA makes use of a symbolic memory space (including the memory of the Ethernet device) and a symbolic register bank to store information about the propagation of network bytes in our system. Each component of this symbolic storage area has a 1:1 correspondence with the real component in the system architecture. The network bytes propagation information is stored in objects called *Expressions*, which can be of several types, for instance, Label, Operation, Predicate, DoubleExpression and QuadExpression.

A Label has a unique integer that identifies it. An Operation is characterized by the operation that is performed in its two operands, which can be any type of Expression. A Predicate is formed by a specific relation comparing an Expression on the left-hand side and another on the right-hand side. A DoubleExpression represents the two Expressions associated with each byte of a 16-bit word, and a QuadExpression is formed by the four Expressions associated with the four bytes of a 32-bit word. DACODA discovers equality predicates every time labeled data or a symbolic expression is explicitly used in a conditional control flow transfer.

We describe how DACODA works with the example given in Figure 4. First, when a network packet is read from the Ethernet device, we create a Label object identified by a unique integer for every byte of the packet. Suppose a byte from a network packet is labeled with "`Label 1832`" when it is read from the Ethernet card. As the byte is stored in the Ethernet device memory, its corresponding label object is stored in the Ethernet device symbolic memory. DACODA then follows the byte through the device into the processor where the kernel reads it into a buffer. In this case wherever the byte is stored, DACODA tracks it down through its Label in the symbolic storage space. Suppose the kernel copies this byte into user space and a user process moves it into the AL register, adds the integer 4 to it, and makes a control flow transfer predicated on the result being equal to 10.

For the first instruction DACODA stores an Expression of type Label (number 1832) into our symbolic AL register. For the second instruction, DACODA creates an Operation object with operation **ADD**, operand1 **Label 1832** and operand2 **Constant 4** and stores this object into our symbolic AL register. The Zero Flag (ZF) is used by the Pentium for indicating equality or inequality. DACODA associates two expressions with ZF, left and right, to store the expressions for the last two data that were compared. ZF can also be set by various arithmetic instructions but only explicit comparison instructions set the left and right pointers in DACODA. Thus, in the third instruction DACODA sets the left expression of ZF to be the expression stored at the AL symbolic register and the right expression to **Constant 10**. If any subsequent instruction checks ZF and finds it to be set, DACODA creates an equality predicate. This is exactly what happens for the last instruction in our example, with DACODA discovering the predicate (in prefix notation),"`(EQUAL (ADD (Label 1832) 4) 10)`". DACODA defines a *strong, explicit equality predicate* to be an equality predicate that is exposed because of an explicit check for equality.

```
1. mov    al,[AddressWithLabel1832]
; AL.expr <– (Label 1832)

2. add    al,4
; AL.expr <– (ADD AL.expr 4)
; AL.expr == (ADD (LABEL 1832) 4)

3. cmp    al,10
; ZF.left <– AL.expr
; ZF.left == (ADD (Label 1832) 4)
; ZF.right <– 10

4. je  JumpTargetIfEqualToTen
; P = Predicate(EQUAL ZF.left ZF.right)
; P == (EQUAL (ADD (Label 1832) 4) 10)
; if (ZF == 1)
;   AddToSetOfKnownPredicates(P);
; Discover predicate if branch taken
```

Fig. 4. DACODA Example

Whenever Minos catches an attack, DACODA provides information about it, such as processes involved, if the attack involved kernel or user processes, tokens that compose the attack trace and the predicates found.

4.2 The Signature Generation Process

We have modified DACODA to detect if the attack contains a constructed bit-string that will exploit the ASN.1 vulnerability. In general there are two pieces of information that need to be placed into the ASN.1 Petri net signature: the headers and the sizes. The headers can be easily generated because they are automatically detected when DACODA finds an equality predicate. The sizes are always in the bytes immediately following the headers so those can be found based on where the equality predicates are. The start place is automatically inserted into the Petri net by searching for the first predicate relative in the ASN.1 attack (0x23). After that when DACODA detects an equality predicate it will be used to generate a new transition. If an equality predicate has already been seen, then the output of the transition will lead back to the original place already generated.

In addition to looking at the equality predicate, DACODA will also keep track of the next four bytes. The justification of this is based on the fact that equality predicates often represent tags or headers of a packet. Then the size (if applicable) will be one or more of the next four bytes. Based on this assumption, the next step is to determine how far the next byte used in an equality predicate is away from this one in terms of a constant and the size bytes that were recorded. The current implementation chooses from a list of possible formulas. Currently there are eight formulas that DACODA can choose from. If A B C and D are the four bytes following the tag they are: A, B, C, D, A+B, A-B, (A<<4)+B -C, other (constant number). The last two formulas are relevant for the ASN.1

Petri net signature. The correct formula can be determined based on how far away the next equality predicate is. In order to track the size, DACODA will add places and transitions to keep track of the length by creating tokens. The places and transitions will be generated based on the formula that was detected. This stored length can be used for comparison and to figure out where other headers would be. The size can be calculated in tracked by Petri nets by adding an extra transition that will produce tokens until a certain result is reached. This concept can be seen in Figure 3 where the number of a's seen gets calculated and then is used to see if the correct number of b's are present. DACODA generates Petri net places and transitions to track the sizes after the size is detected from the list of formulas. Closely tracking the sizes and headers accounts for the control flow and data flow sensitivity and will eliminate many false positives. The execution point at which Minos stops the attack represents the attack place. If a token reaches the final place than an attack occurred.

The Petri net outputted is consistent with its formal definition [11]. Figure 5 represents the formal definition of the Petri net signature for the ASN.1 double free vulnerability. The main progression of a token from the initial place to the attack place will basically proceed down the main horizontal string of places and transitions. Place 1 is the unlabeled Place where the token starts off and Place 7 is when an attack is detected. The transitions from Place 1 to Place 2 and Place 2 to Place 3 are straight forward. The token simply transitions after a constructed bitstring tag and a nested bitstring tag are found. After we have found the first bit string we need to take into account both the number of bytes and the number of unused bytes. Places 3 and 4 both create new tokens into Places 8 and 9 respectively. In order to keep track of the fact that there may be unused bytes in a bitstring, the Petri net will subtract the tokens from Place 8 for every unused byte counted in Place 9. However, these tokens are transitioned to Place 11. Place 5 will transition to the next header in the ASN.1 message format using the tokens that are still remaining in Place 8. These tokens from

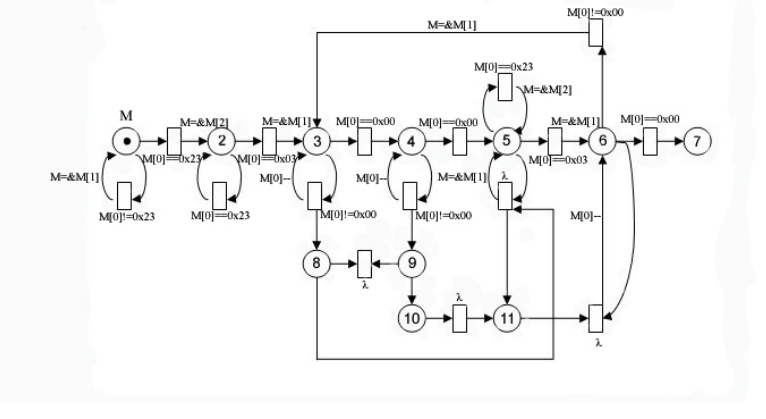


Fig. 5. Petri Net Graph for the ASN.1 Vulnerability Signature Format

the used bytes in the bitstring will rejoin the tokens from the unused bytes in Place 11. We will then transition into Place 6 when we find the next bitstring and the sizes can be compared using the tokens from Place 11. If the sizes are equal than we transition into Place 7 and an attack is detected. This formal definition can be transformed into a signature checker using a program that will interpret the Petri net.

5 Evaluation

In this section we evaluate our proposed Petri Net class of double-free vulnerability signature. First, we show that previous approaches [3] are not effective for vulnerabilities such as the ASN.1 Double-Free addressed in this paper. The three classes of vulnerability signatures proposed in a recent work [3] are regular expressions, symbolic constraint notation, and Turing machines. Since the signature generation techniques used in [3] are not available to us, the vulnerability signature for each class had to be manually created. It is important to point out that since manual creation was utilized, the vulnerability signatures are more specific and efficient (yield better results in false positives, false negatives, and time overhead) than if the signatures had been automatically created.

Looking at the properties of the ASN.1 vulnerability implies that the vulnerability signature must be based on the constructed bitstring headers. The only difference between an exploit message and a regular message will be the order of bitstrings and constructed bitstrings. But more information about the headers, such as the length of bitstrings, can be added to reduce false positives. A list of properties that must be true to produce an attack is seen below where 0xXX indicates that the byte value can be anything and 0xNN, 0xPP, 0xQQ, etc. indicates that the value of the byte should be N, P, Q, etc. respectively: (i) The pattern must begin with one or more constructed bitstring headers: 0x23(tag) 0xXX(size) or 0x23(tag) 0x80(tag) 0xXX(size) 0xXX(size), (ii) Afterwards there can be one or more bitstring headers followed by the bitstring: 0x03(tag) 0xLL(size) 0xMM(unused bits) $0xXX^{L-M}$(bitstring data) or 0x03(tag) 0x80(tag) 0xLL(size) 0xMM(size) 0xKK(unused bits) $0xXX^{(L<<4)+M-K}$ where the last bitstring's size is not zero, L-M refers to the mathematical result of L - M and (L<<4) +M - K refers to the binary representation of L shifted 4 bits to the left then that result added to M-K, (iii) There must be one or more constructed bitstring headers: 0x23(tag) 0xXX(size)or 0x23(tag) 0x80(tag) 0xXX(size) 0xXX(size) and (iv) The second of the two consecutive bitstrings should appear: 0x03(tag) 0xLL(size) 0xMM(unused bits) $0xXX^{L-M}$(bitstring data) or 0x03(tag) 0x80(tag) 0xLL(size) 0xMM(size) 0xKK(unused bits) $0xXX^{(L<<4)+M-K}$ where the last bitstring's size is not zero.

Regular expressions are perhaps the most widely used pattern recognition scheme for its power and efficiency. However, in the case of describing the ASN.1 vulnerability signature it cannot distinguish between some valid traces and malicious traces. The reason is that regular expressions do not take into account bitstring length and may misinterpret bytes. The content of the bitstrings can

```
[0x23] [*]{1, 65535} [0x03] [^0][*] {1, 65535} [0x23] [*]{1, 65535} [0x03] [^0][*] {1, 65535}
```

Fig. 6. Single Regular Expression for the ASN.1 Vulnerability

```
(inp[X] = 0x23) ^
{[(inp[X+2] = 0x03) ^ (inp[X+3] = 0x01) ^ (inp[X+3] = 0x00) ^
 (inp[X+6] = 0x23) ^ {(inp[X+8] == 0x03) ^ (inp[X+9] != 0x00)}] ˇ
[(inp[X+2] = 0x03) ^ (inp[X+3] = 0x02) ^ (inp[X+3] = 0x00) ^
(inp[X+7] = 0x23) ^ {(inp[X+9] == 0x03) ^ (inp[X+10] != 0x00)}] ˇ
[(inp[X+2] = 0x03) ^ (inp[X+3] = 0x03) ^ (inp[X+3] = 0x00) ^
 (inp[X+8] = 0x23) ^ {(inp[X+10] == 0x03) ^ (inp[X+11] != 0x00)}] ˇ
[(inp[X+2] = 0x03) ^ (inp[X+3] = 0x04) ^ (inp[X+3] = 0x00) ^
 (inp[X+9] = 0x23) ^ {(inp[X+10] == 0x03) ^ (inp[X+11] != 0x00)}] ˇ
. . .}
```

Fig. 7. Symbolic Constraint Signature for the ASN.1 Vulnerability

always have sequences that look like headers, but regular expressions will not be able to determine if it is a header or if it is data. As a result, signatures created for the ASN.1 vulnerability that do not produce false negatives using regular expressions will produce false positives and those that do not produce false positives have false negatives. For the purposes of this experiment, in order to show that the ASN.1 attack could be detected by vulnerability signatures it is important to eliminate false negatives and we did so by manually creating the ASN.1 regular expression (Figure 6).

The Symbolic Constraint signature class is represented by a set of Boolean formulas. The power of these signatures over regular expressions is that it has the power to specify a range of input bytes or any specific input byte for comparison with a value. However, they do not have the capability to handle loops. Due to the recursive nature of the ASN.1 constructed bitstring structure, loops are necessary in the signature. Although approximations can be made in the symbolic constraint notation, any reasonable signature would be extremely long. The symbolic constraint signature seen in Figure 7 is very drawn out, but it is still not enough to deal with the complexity of the ASN.1 vulnerability. This symbolic constraint signature will look for a constructed bitstring. Then within that constructed bitstring it will look for a bitstring of length n and then look for another constructed bitstring and a bitstring of length k afterwards. Then numbers n and k are varied from 1 to 128 to keep the length down but still there are 16,384 combinations. Of course if the bitstrings have a longer length they will still not be detected by this symbolic constraint signature.

Turing machines will of course be able to represent the ASN.1 vulnerability signature because the full ASN.1 constructed bitstring decoder function can simply be emulated. However, Turing machines are complex, inefficient, and will not always terminate.

Second, we analyze the effectiveness of our Petri Net approach along with regular expressions, symbolic constraint, Turing machines with regard to false positives, false negatives, and complexity. The results are presented in Tables 1 and 2.

Table 1. Four Vulnerability Signatures Classifying Randomly Generated ASN.1 Attack Packets

Signature Class	Attacks	False Negatives	Valid Traces	Seconds
ASN.1 Daemon	887965	0	112035	305
Regular Expressions	1000000	0	0	17
Symbolic Constraints	71691	928309	0	144
Turing Machines	887965	0	112035	72
Petri Nets	1000000	0	0	71

In order to test for false negatives for each signature class a random ASN.1 constructed bitstring generator was developed to create a large number of bitstrings that will generate an attack. Each length in the constructed bitstring uses a random number generator to construct one million attack traces. Each attack trace was specifically generated to contain bitstrings and constructed bitstrings in an order that would constitute an attack. However, zero length bitstrings were allowed so not all of the traces would constitute an attack. Using a specially designed ASN.1 daemon modeled after the Windows ASN1BERDecBitstring function [7], each of the 1,000,000 attack traces was tested to see if it would cause a double free attack. About 88% of the bitstrings were determined to cause an attack. The other 12% would not result in an attack.

The Turing machine signature returned the best results because it has no false negatives. Regular expressions and Petri nets are both able to detect all the attack packets, but the valid packets due to zero length strings are also detected creating false positives. Fortunately, Petri net signatures can eliminate this type of error by adding an extra place and a lambda transition. However, our generated Petri net signature is not yet capable of producing this result. The regular expression used in this paper was specifically constructed to achieve zero false negatives. Due to the limitations of regular expressions this construction will produce many more false positives (Table 2). However, symbolic constraint signature is limited in detecting attack packets. Only 7% of the packets were found, so about 93% of potential attacks were missed. The recursive nature of the ASN.1 Windows vulnerability is the reason that symbolic constraint signatures yield worse results than regular expressions. For simpler vulnerabilities symbolic constraints will perform better. In order to rectify this problem, an even longer symbolic constraint signature would need to be developed. However, this will also increase the time overhead.

Table 2 contains the results when running the signatures against real traffic data. Each signature was used to check one gigabyte of trace data to search for any matching patterns. The TCP dump that was used represents real traffic from a research institute in Taiwan and contains about 4.8 million traces of which about 1.4 million had the ASN.1 tag present. The traffic data does not contain any attacks so any matching trace is considered a false positive. As expected the regular expression signature is much too general and returns a very large number of false positives. The high number of false positives for regular expressions is due to the lack of knowledge of size. It will match traces that have the 0x23

Table 2. Four Vulnerability Signatures Classifying Real Traffic Data

Signature Class	False Positives	Seconds
Regular Expressions	167432	934
Symbolic Constraints	7	967
Turing Machines	0	1083
Petri Nets	3	917

tag at the beginning and the rest of the pattern at the end regardless of the size bits. The results for symbolic constraint signatures and Petri nets yielded a small number of false positives with an execution time that was about 11-15% faster than the Turing machine signature scheme. The three false positives detected by Petri nets are all contained within the seven false positives detected by symbolic constraint notation. Those seven false positives were also part of the 167,432 false positives detected by the regular expression signature. Although Turing machine signature matching was clearly slower, it did have the advantage of zero false positives.

One of the traces that was matched as an attack by Petri net, symbolic constraint, and regular expressions is displayed in Figure 8. The bolded part is the sequence in the trace that is causing the false positive. This particular trace is a TCP packet found in the trace file. Although this sequence does closely resemble an attack, it is correctly identified by Turing machines because the initial constructed bitstring has a length of only two bits. Therefore none of the following sequences would actually be analyzed by the vulnerable Windows ASN.1 decoding function. This extra size constraint was not automatically generated by a Petri net signature and is too complex for symbolic constraint and regular expression signatures. It is unclear whether the automatically generated Petri net signature can be expanded to also track the size of each constructed bitstring to eliminate this type of false positive. As a result, a valid trace will be classified as an attack. However, this pattern did not occur often in the large number of test cases where the false positive rate has been shown to be much less than 1% for Petri nets.

The results in these tests show the inadequacy of regular expression and symbolic constraint signatures due to false positives and false negatives respectively. Turing machines have been used in this example to produce the best results, but they still have the problem of becoming undecidable. According to [3], Turing machines will not terminate for certain signatures. In addition, the automatic generation of Turing machine signatures will be produced using low level instruction code generated from the binary which would be very hard to verify or debug. Conversely, Petri nets are naturally represented as a graph which is very easy to interpret meaning from. Test results showed that Petri nets are capable of accurately represent the complex ASN.1 vulnerability signature producing no false negatives and a very small number of false positives (less than 1%). The time overhead of the Petri net signature was also significantly less than Turing machines. In addition, Petri net signatures can be automatically generated with a symbolic execution tool like DACODA. Generation and verification is a very

```
11 0A 99 F8 C0 23 02 03 03 01 01 23 23 07 46 42
47 45 42 45 4A 45 50 43 41 43 41 43 41 43 41 43
41 43 41 43 41 43 41 43 41 43 41 43 41 41 41 00
20 46 48 45 50 46 43 45 4C 45 48 46 43 45 50 46
46 46 41 43 41 43 41 43 41 43 41 43 41 43 41 42
4E 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 11 00 00 07 00 00 00 00 00 00 00 00 00 E8
03 00 00 00 00 00 00 00 00 07 00 56 00 03 00 01
00 01 00 02 00 18 00 5C 4D 41 49 4C 53 4C 4F 54
5C 42 52 4F 57 53 45 00 02 00 56 41 49 4F 00
```

Fig. 8. Example of False Positive

important aspect of vulnerability signatures because it is necessary that they are implemented quickly. One other comparison factor for the four signatures is the signature size. The longer the signature the more space overhead it will require. Also more time overhead will be needed to process it. Regular expressions produced the smallest signature, 80 bytes, whereas, symbolic constraint notation produced the largest signature, 123 kilobytes. The sizes of the Petri net signature, 1.8 kilobytes, and the Turing machine signature, 3.2 kilobytes, were pretty similar. Consequently, Petri net vulnerability signatures performed best overall for the four properties of an ideal signature.

Other double free vulnerabilities can also be detected by tracking the equality predicates in DACODA. The simpler attacks will not need to take into account the size of bitstrings like the ASN.1 double free vulnerability signature did. In the cases where the vulnerability is not data flow sensitive the length in between predicates can just be set to a constant instead of an expression. Conversely, for other complex attacks, the formulas used for the size may need to be expanded to detect other kinds of attacks. Buffer overflow vulnerabilities can also be detected by taking the distance in between predicates into account. Other vulnerabilities, like format string vulnerabilities, should be able to be detected simply using the equality predicates and rules already set in DACODA because the critical properties, like the format inconsistency, will be detected.

6 Related Work

Current defense against malicious activity can be split up into two types of intrusion detection systems: host based and network based.

Host based systems [4, 5, 9, 13, 15, 19] have the distinct advantage over network based systems of being able to utilize all of the information that exists on the host, making detection much more feasible because of this extra information. However, preventing attacks at the network level would cause less congestion on the network and save valuable host resources for useful work. In addition, due to the many hosts that may be on a network, deploying intrusion detection is more feasible if it only needs to be done once to protect the entire network.

Network intrusion detection systems traditionally rely on exploit-based signatures [12, 16, 18, 20, 23]. The exploit based signatures for attacks like buffer overflows, are most commonly the no-op sleds or malicious shell codes of the exploit. However, due to register springs and polymorphism, exploit signatures may not be the best alternative because of the great number of different signatures required [6]. One other variation of a NIDS utilizes network emulation to execute the possible paths for every trace [14]. However, running the program at the network level produces a considerable delay and is limited by only detecting single packet attacks.

7 Conclusions and Future Work

Although security in software programs has improved, current attacks have become more complicated and harder to detect using traditional exploit-based signatures. The ASN.1 vulnerability is a great example of an advanced double free vulnerability that is very hard to detect due to its control flow and data flow sensitive nature. Vulnerability signatures are a new method of detecting intrusions that uses properties of the vulnerability that must be prevalent in all possible exploits of that vulnerability. Petri nets were introduced as a new class of vulnerability signature that can be used to detect not only double-free but also other types of vulnerabilities that can be expressed or caught using regular expressions. They are a very efficient, concise, and effective way of describing signatures (both vulnerability and exploit). They are more powerful than regular expressions and still efficient enough to be practical. In order to test this new class, it was analyzed along side three other classes of vulnerability signatures presented in the literature [3] in relation to the Windows ASN.1 vulnerability. The results of the Petri net class were very promising (even though they have only been shown to work against a single vulnerability) due to the very low false positive rate and 0% false negative rate. Only Turing machines provided a better identification rate. However, Petri nets can be automatically generated using a symbolic execution tool like DACODA [6] and are very easily represented as a graph making it easier to understand and debug. Turing Machines also have a distinct disadvantage of time overhead. Testing the effectiveness of Petri net signatures on other vulnerabilities is left for future research although the same methodology should be applicable for other vulnerabilities.

References

1. Biba, K.J.: Integrity Considerations for Secure Computer Systems. In: MITRE Technical Report TR-3153 (April 1977)
2. Bishop, M.: Computer Security: Art and Science (2003)
3. Brumley, D., Newsome, J., Song, D., Wang, H., Jha, S.: Towards Automatic Generation of Vulnerability-Based Signatures. In: IEEE Symposium on Security and Privacy (May 2006)

4. Costa, M., Crowcroft, J., Castro, M., Rowstron, A., Zhou, L., Zhang, L., Barham, P.: Vigilante: End-to-end containment of Internet worms. In: SOSP 2005: Proceedings of the twentieth ACM Symposium on Operating Systems Principles, pp. 133–147. ACM Press, New York (2005)
5. Crandall, J.R., Chong, F.T.: Minos: Control Data Attack Prevention Orthogonal to Memory Model. MICRO, 221–232 (December 2004)
6. Crandall, J.R., Su, Z., Wu, S.F., Chong, F.T.: On Deriving Unknown Vulnerabilities from Zero-Day Polymorphic and Metamorphic Worm Exploits. ACM CCS, 235–248 (November 2005)
7. Eclipse, S.: kill-bill windows exploit, http://www.phreedom.org/solar/exploits/msasn1-bitstring/kill-bill.tar.gz
8. King, J.C.: Symbolic execution and program testing. Commun. ACM 19(7), 385–394 (1976)
9. Kiriansky, V., Bruening, D., Amarasinghe, S.: Secure Execution Via Program Shepherding. In: USENIX, pp. 191–206 (2002)
10. Larmouth, J.: Asn.1 complete. open system solutions (1999)
11. Murata, T.: Petri Nets: Properties, Analysis, and Applications. Proceedings of the IEEE 77(4) (April 1989)
12. Newsome, J., Karp, B., Song, D.: Polygraph: Automatically generating signatures for polymorphic worms. In: Proceedings of the IEEE Symposium on Security and Privacy, pp. 226–241 (2005)
13. Newsome, J., Song, D.: Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software. In: Proceedings of the 12th Annual Network and Distributed System Security Symposium (NDSS 2005) (Febuary 2005)
14. Polychronakis, M., Anagnostakis, K., Markatos, E.: Network-level polymorphic shellcode detection using emulation. Institute for infocomm research, singapore (2005)
15. Qin, F., Wang, C., Li, Z., Kim, H.-S., Zhou, Y., Wu, Y.: LIFT: A Low-Overhead Practical Information Flow Tracking System for Detecting Security Attacks. MICRO-39, 135–148 (December 2006)
16. Singh, S., Estan, C., Varghese, G., Savage, S.: Automated worm fingerprinting. In: OSDI (2004)
17. Szor, P.: The Art of Computer Virus Research and Defense (2005)
18. Tang, Y., Chen, S.: Defending Against Internet Worms: A Signature-based Approach. In: INFOCOM (2005)
19. Vachharajani, N., Bridges, M.J., Chang, J., Rangan, R., Ottoni, G., Blome, J.A., Reis, G.A., Vachharajani, M., August, D.I.: Rifle: An architectural framework for user-centric information-flow security. In: Proceedings of the 37th International Symposium on Microarchitecture (MICRO), December 2004, pp. 39–58 (2004)
20. Wang, K., Stolfo, S.: Anomalous Payload-Based Network Intrusion Detection. In: Jonsson, E., Valdes, A., Almgren, M. (eds.) RAID 2004. LNCS, vol. 3224, pp. 201–222. Springer, Heidelberg (2004)
21. Wikipedia. Wikipedia. Petri net, http://en.wikipedia.org/wiki/Main_Page
22. eEye advisory for AD20040210-2, http://www.eeye.com
23. SNORT: The open source network intrusion detection system (2002), http://www.snort.org

Distinguishing between FE and DDoS Using Randomness Check*

Hyundo Park[1], Peng Li[2], Debin Gao[2], Heejo Lee[1], and Robert H. Deng[2]

[1] Korea University, Seoul, Korea
{hyundo95,heejo}@korea.ac.kr
[2] School of Information Systems, Singapore Management University, Singapore
{pengli,dbgao,robertdeng}@smu.edu.sg

Abstract. Threads posed by Distributed Denial of Service (DDoS) attacks are becoming more serious day by day. Accurately detecting DDoS becomes an important and necessary step in securing a computer network. However, Flash Event (FE), which is created by legitimate requests, shares very similar characteristics with DDoS in many aspects and makes it hard to be distinguished from DDoS attacks. In this paper, we propose a simple yet effective mechanism called FDD (FE and DDoS Distinguisher) to distinguish FE and DDoS. To the best of our knowledge, this is the first effective and practical mechanism that distinguishes FE and DDoS attacks. Our trace-driven evaluation shows that FDD distinguishes between FE and DDoS attacks accurately and efficiently by utilizing only memory of a very small size, making it possible to be implemented on high-speed networking devices.

Keywords: Network Security, Distributed Denial of Service, Flash Event, Randomness Check.

1 Introduction

Flash crowd was used to refer to the situation when thousands of people went back in time to see historical events anew [16,11]. We use the term *Flash Event (FE)* to refer to a similar situation in which a large number of users simultaneously access a computer server. The computer server that experiences very high load during the event could be a popular web server, e.g., the official Olympic web site during the Olympic games, a course registration server at the beginning of a school semester, and etc. The most important characteristics of Flash

* This research was supported by the MIC, Korea, under the ITRC support program supervised by the IITA(IITA-2008-(C1090-0801-0016)), the IT R&D program of MKE/IITA(2008-S-026-01) and partially supported by Defense Acquisition Program Administration and Agency for Defense Development under the contract(2008-SW-51-IM-02).

T.-C. Wu et al. (Eds.): ISC 2008, LNCS 5222, pp. 131–145, 2008.

Event (FE) include that it is created by legitimate users, and that the server experiences abnormal high demand.

Denial of Service (DoS) attacks attempt to make a computer resource unavailable to its intended users. A very common method of the attack involves saturating the victim machine with external communication requests such that it cannot respond to legitimate traffic. Moreover, *Distributed Denial of Service (DDoS)* attacks attempt to do so by sending these external requests from many compromised machines (a.k.a. zombies, daemons, agents, slaves, etc.) distributed around the world. DoS attacks have become a major threat to network security in the past few years. Over 12,000 worldwide DoS attacks had been observed over three weeks in 2001 [5]. DDoS attack was listed as the most financially expensive security accident on the 2004 CSI/FBI Computer Crime and Security Survey [14]. Many DDoS attacks bring down the victim server by consuming a lot of resources on the victim machine, e.g., CPU resources, memory resources, and bandwidth. Some attacks spoof the source IP addresses, e.g., SYN flood, while others do not. The former is relatively easy to detect because an ACK never comes back from the client; while zombies that send requests to the victim server like what legitimate users would do can be much harder to detect. Despite a lot of research done in the past few years, accurately detecting DDoS remains a hard problem.

To make the problem even more difficult, FE and DDoS attacks share many similar characteristics, and are very difficult to distinguish from one another [11]. Both FE and DDoS are caused by a large number of client requests. The consequences of both FE and DDoS include slow responses and connection drops. In the case of FE and DDoS where source IP addresses are not spoofed, what makes FE and DDoS attacks different is user intention, which is hard to detect by the victim server. Although FE and DDoS share similar characteristics and are hard to tell from one another, it is of great interest to be able to distinguish them, because very different actions need to be done in rectifying these two events. In the case of a FE, the server administrator may want to quickly enable or increase the number of CDNs (Content Distribution Networks), load sharing mechanisms, and etc. so that more users can be accommodated. In the case of a DDoS attack, the server administrator may want to quickly deploy/enable filters at the border gateway to filter out attack traffic so that legitimate requests are not dropped.

In this paper, we propose a mechanism called FDD to distinguish between FE and DDoS attacks using randomness check. To the best of our knowledge, this is the first effective and practical approach that distinguishes FE and DDoS attacks. Studies have found that a noticeable difference between FE and DDoS is in the distribution of clients and of their requests. During FE, a large number of *clusters* active during an FE had also visited the sites before the event [11]. A *cluster* is a group of clients that are close together topologically and are likely to be under a common administrative control [4]. Since clusters sending a large number of requests to the server can be frequently observed during FE, the source addresses of requests received by the server during a FE is not random.

In other words, these source addresses are predictable and follow some probability distribution which can be approximated by observing previous requests and analyzing their source addresses. However, DDoS does not share this unique feature. DDoS is usually due to an increase in the number of clients or a particular client sending requests at a high rate. Client distribution across ISPs (clusters) does not follow population distribution [11]. Intuitively, this means that the source addresses of an DDoS attack are much less predictable, and look more like random addresses to the victim server.

FDD distinguishes between FE and DDoS using randomness check of the distribution of clients among clusters. In order to do this, we construct a matrix to capture the cluster distribution, and apply two operations, XOR and AND, to our matrices to remove or keep, respectively, the overlapping clusters. Simply put, the matrix is able to capture the randomness feature in the client distribution, which is a key feature to distinguish FE and DDoS. The XOR and AND operations further enables us to capture the dependency between current requests and requests received previously, which not only further improves the accuracy of the randomness check, but enable us to distinguish source-spoofed DDoS and non-source-spoofed DDoS. With trace-driven evaluations, we show that the proposed matrix operations and randomness check are able to accurately distinguish between FE and DDoS attacks.

In the rest of the paper, Section 2 describes characteristics of FE and DDoS, as well as related work. Our approach of distinguishing between FE and DDoS is presented in Section 3. In Section 4, we present evaluation results. Finally, Section 5 concludes our paper.

2 Characteristics of FE and DDoS and Related Work

2.1 FE and DDoS

In this subsection, we explain some characteristics of FE and DDoS as found by Jung et al. [11]. FE and DDoS attacks are similar to each other in many aspects. For example, when the server is in a Flash Event or under a DDoS attack, the traffic volume would be considerably high, resulting in slow responses and dropping of connections. On the other hand, FE and DDoS attack are different in many ways. One aspect in which they differ most is the distribution of distinct clients among clusters, which are constructed by the network-aware client clustering technique [4]. During FE, the number of distinct clusters is much smaller than that of distinct clients. However, during DDoS attacks, these two numbers become very close [11]. In other words, the distribution of requests among clusters in FE is very different from that in DDoS attacks. One of the most important reasons is because DDoS attacks usually make use of vulnerable machines on the Internet, and these vulnerable machines are distributed randomly. Below we summarize some of the important characteristics of FE and DDoS [11] which we explore to design FDD.

First, the number of requests sent to the server would increase dramatically during both FE and DDoS attacks. We focus on the most common DDoS

attacks — flooding attacks with a big traffic volume that consume various resources at the victim.

Second, the number of distinct clusters during the FE is much smaller than the number of distinct clients. However, DDoS requests come from clients widely distributed across clusters in the Internet.

Third, a large number of clusters active during an FE had also visited the sites before the event. A possible explanation is that many clusters would have at least one client that accessed the site already when the overall request rate is high. However, in the case of DDoS, an overwhelming majority of the client clusters that generate requests are new clusters not seen by the site before the attack.

2.2 Related Work

Varieties of DDoS detecting mechanisms have been proposed in the literature. He et al. proposed a mechanism to detect SYN flooding attack using Bloom filter [19]. They update the client list with a Bloom filter; if a SYN request shows up on the network, they increase the corresponding counter for this client in the list; but if a SYN/ACK request comes from the same client, they decrease the number of the same counter by one. Using this method, they can detect SYN flooding attacks by checking the counters on the list. Another mechanism, proposed by Wang et al. [10], make use of the ratio of the numbers of SYN and FIN/RST. During a SYN flooding attack, there would be a significant amount of SYN packets, but the number of FIN/RST packets would not be as large as that of SYN packets. However, these two mechanisms only react on TCP protocol and will recognize FE and DDoS attacks as the same type of event. The mechanism proposed by Peng et al. detects DDoS attacks by monitoring the distribution of source IP addresses [18]. The number of new IP addresses that have not been observed before is very large during DDoS attacks. With a hash function, they check the distribution of those new IP addresses to detect DDoS. However, this approach would fail if the attackers were not to spoof the IP addresses. Feinstein et al. develop a statistical approach to detect DDoS attack [15]. They exploit the characteristic that the distribution of source IP addresses during DDoS attacks is uniform, and detect DDoS attacks with the help of Chi-square statistics and entropy. Their experiment shows that the Chi-square value would increase dramatically during DDoS attacks. However, the threshold of their detecting system depends on the statistical results and need to be changed in different network environments.

In other category to counter with DDoS attacks, several protection mechanisms have been proposed recently. Stavrou et al. proposed a mechanism to counter DoS attacks, routing to send each packet through a randomly selected overlay node, with stateless protocol for authenticating users to the infrastructure and an efficient per-packet authentication scheme [3]. A new network-based flood protection scheme is proposed by Casado et al. which is called CAT(Cookies Along Trush-boundaries) [17]. The scheme uses flow cookies and IP black-list lookup which is deployed between a cookie box and a web server. AITF(Active

Internet Traffic Filtering) proposed by Argyraki et al. leverages the recorded route information to block attack traffic [12]. If an attacker spoofs his/her source IP address, attack packets with multiple source IP addresses will have one routing path and AITF can block the attack packets close to the attack source.

The majority approaches focus on deal with DDoS attacks or abnormal situation of traffics without considering a FE. Even though some approaches take account of a FE, they set a FE as one of abnormal activities without distinguishing from DDoS attacks. Since a FE is caused by legitimate users, the countermeasure of server administrator during a FE is very different from it during DDoS attacks. In this paper, we propose a new mechanism to distinguish between FE and DDoS attacks.

3 FDD (FE and DDoS Distinguisher) Using Randomness Check

In this section, we first provide an overview of FDD, and then explain the matrix construction and operations, and the randomness check in details.

3.1 Overview of FDD

Our approach, FDD (FE and DDoS Distinguisher), is designed to distinguish between FE and DDoS attacks using a matrix construction, two newly defined matrix operations, and the randomness check on the matrices. We first motivate the idea of grouping incoming requests into clusters. The most important reason why we group requests into clusters is that it enables us to draw a line between FE and DDoS. As briefly mentioned in Section 1, a cluster is a group of clients that are close together topologically and are likely to be under a common administrative control [4]. It has been found that the number of distinct clusters during the FE is much smaller than the number of distinct clients, whereas DDoS requests come from clients widely distributed across clusters in the Internet [11]. Since clustering incoming requests can help distinguishing FE and DDoS, we take it as the first step in FDD.

To motivate the idea of performing randomness check, we analyze another nice distinction between FE and DDoS that a large number of clusters active during an FE had also visited the sites before the event, whereas an overwhelming majority of the client clusters that generate DDoS requests are new clusters not seen by the site before the attack [11]. This serves as a clue to distinguish between FE and DDoS attacks, if we can measure the extent to which clusters overlap in the incoming requests. Comparing this cluster overlapping along the time axis with the property of randomness [2], in which all elements in a sequence should be generated independently from one another, and the value of the next element in the sequence cannot be predicted, we can see some similarity in the two — if cluster overlap happens frequently in incoming requests, we would be able to predict the cluster from which the next request comes with certain (non-negligible) probability. This is why randomness check can help distinguishing

FE and DDoS attacks. In order to do the randomness check on the cluster distribution of incoming requests, we first capture the cluster distribution using a matrix for each time unit (could be 1 second, 1 minute, etc. depending on the actual working environment), and then check whether the clients are randomly distributed or not by checking randomness of the matrix.

What we describe in the previous paragraph motivates the idea of doing randomness check, but does not solve the problem of representing cluster overlaps. FDD introduces two matrix operations in order to represent cluster overlapping: the XOR operation between the cluster matrix for the current time unit and the one for the previous time unit to remove the overlapping clusters, and the AND operation between two matrices to remove all but the overlapping clusters.

Having motivated the idea of using matrices, checking randomness and the two matrix operations to capture cluster overlap, we now describe the steps involved for FDD to distinguish FE and DDoS.

1. Construct the matrix to represent cluster distribution;
2. Apply XOR and AND operations between matrices of the current and the previous time units;
3. Check the randomness.

3.2 Matrix Construction and Operations

Matrix construction is to capture the cluster distribution of incoming requests in a matrix. We try to make this construction as simple as possible, so that it can possibly be implemented on, e.g., high-speed routers. Note that many other clustering methods could be used, e.g., the approach by Krishnamurthy and Wang [4]. Here, we present a very simple technique. We first divide each IP address into four octets as presented in the following notation, where the length of each octet is one byte.

$$\mathrm{IP}_1.\mathrm{IP}_2.\mathrm{IP}_3.\mathrm{IP}_4 \tag{1}$$

In this simple technique, we simply map an incoming request to a specific location in the matrix determined by IP_2 (used as the column index of the matrix) and IP_3 (used as the row index of the matrix) of the IP address. That is,

$$i = \mathrm{IP}_3 \qquad \text{and} \qquad j = \mathrm{IP}_2 \tag{2}$$

where i is the row index and j is the column index of the matrix. A very nice property of this simple approach is that it results in a matrix of fixed size, which is a property that many other clustering mechanisms do not have. Note that we use this very simple method of constructing the matrix to demonstrate the idea of using randomness check. Other more sophisticated techniques for constructing a matrix, e.g., by making use of a clustering mechanism [4], could be used as well for better representation of the incoming traffic. We address this in our future work.

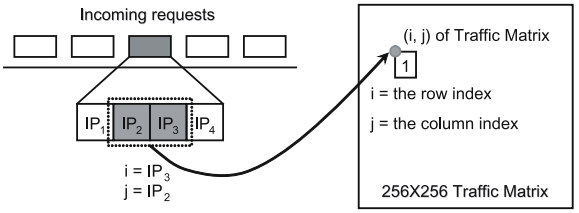


Fig. 1. Place of an incoming request in matrix construction

Fig. 1 illustrates how an entry of the matrix is located based on the IP address of the incoming request. The matrix is initialized with zeros in all entries. For each incoming request and the corresponding entry located based on the IP address, we overwrite the content of the entry with the value 1. As a result, we have a matrix of the size 256×256 and each cell in the matrix contains a binary number. This makes our system very memory space efficient (65,536 bits).

Let M_t denote the matrix constructed by processing incoming requests during the t^{th} time unit. We introduce two operations between matrices constructed at consecutive time units. We define (M_t XOR M_{t-1}) to be a matrix in which each entry is calculated as the XOR (exclusive-or) of the corresponding entries in M_t and M_{t-1}, and (M_t AND M_{t-1}) to be a matrix in which each entry is calculated as the AND of the corresponding entries in M_t and M_{t-1}. Intuitively, the XOR operation between M_t and M_{t-1} is to remove the overlapping clusters, since $1 \oplus 1 = 0$; while the AND operation between M_t and M_{t-1} is to remove all but the overlapping clusters, since $1 \cdot 1 = 1$.

3.3 Matrix Rank as the Randomness Check

Many approaches have been suggested for testing randomness, out of which checking the linear-dependency among fixed-length substrings of its original sequence is a very efficient one. In order to check the linear-dependency among rows and columns of a matrix, the rank of matrix can be used [6]. Diehard [7] is a good example which is widely used for testing the quality of a random number generator. Another application of randomness check using matrix rank is a worm detection algorithm [9] that detects unknown worms by measuring the randomness of the distribution of destination addresses.

We use $R(M_t)$ to denote the rank value of M_t, and define

$$R_{\text{XOR}}(M_t) = R(M_t \text{ XOR } M_{t-1}) \tag{3}$$

$$R_{\text{AND}}(M_t) = R(M_t \text{ AND } M_{t-1}) \tag{4}$$

To obtain the rank of the matrix, we calculate the number of non-zero rows after applying Gaussian elimination. In other words, the rank of the matrix is the number of leading 1's in the matrix.

We can show mathematically why the rank value of a matrix provides a reliable indication of randomness. Given the rank value of r, a $m \times n$ matrix has the following probability of being random

Table 1. Distinguishing FE and DDoS

	$R(M_t)$	$R_{\text{XOR}}(M_t)$	$R_{\text{AND}}(M_t)$
Normal	small	small	small
FE	Medium$^+$	Medium	Medium$^-$
DDoS with spoofed source IP	Large ($> T$)	Large ($> T$)	Small
DDoS without spoofed source IP	Large ($> T$)	Small	Large ($> T$)

$$2^{r(n+m-r)-nm} \prod_{i=0}^{r-1} \frac{\left(1-2^{i-n}\right)\left(1-2^{i-m}\right)}{\left(1-2^{i-r}\right)} \tag{5}$$

where $r = 1, 2, \ldots, \min(m, n)$ [6]. (Eq. 5 is also used for calculating the threshold T of the rank values; see the next subsection and Table 1). For example, a 256×256 matrix has a probability of over 99.999% being random if the rank of it is 252 according to Eq. 5.

3.4 Distinguishing FE and DDoS

In this subsection, we describe how we use the matrix operations and rank measurement to distinguish FE and DDoS (refer to Table 1).

In an FE, the three rank values differ from each other by some noticeable amount, but none of them is very large although the incoming traffic volume could be very big. This is mainly due to the fact that the large amount of requests come from a small number of clusters, which makes $R(M_t)$ not very large, and that there is a certain cluster overlap, which makes both $R_{\text{XOR}}(M_t)$ and $R_{\text{AND}}(M_t)$ slight less than $R(M_t)$ (The superscript $+$ and $-$ indicate slightly larger and less than Medium).

When the server is under DDoS attacks, $R(M_t)$ will be large (and greater than a threshold T; refer to Appendix B for details in calculating T) as the large amount of incoming requests belong to a large number of clusters. In the case of source-spoofed DDoS, cluster overlap is relatively small as the source IPs are chosen randomly. Therefore $R_{\text{XOR}}(M_t)$ will be large (and greater than a threshold T) and $R_{\text{AND}}(M_t)$ will be small. In the case of non-source-spoofed DDoS, cluster overlap is relatively big, and therefore $R_{\text{XOR}}(M_t)$ will be small and $R_{\text{AND}}(M_t)$ will be large (and greater than a threshold T). In this way, the rank of a matrix can be used to determine the randomness of the element distribution on the matrix. Appendix B shows the details in calculating these thresholds.

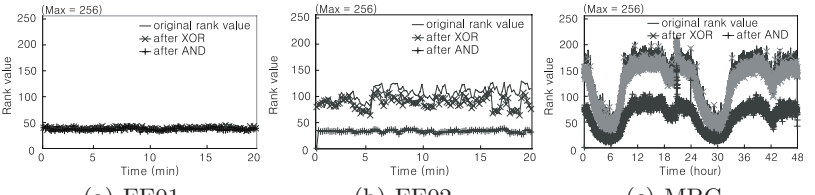
4 Evaluation and Discussion

In this Section, we apply FDD to distinguish between FE and DDoS with traces we obtain from production servers and routers. We first briefly explain the source and the nature of the traces we use, and then present the results when we apply FDD on these traces.

4.1 FE and DDoS Traces

We use two traces that contain FEs. One of them is from the web server of the biggest private broadcast company in Korea, MBC. It contains the web logs for two days, Sep 11th 2004 and Sep 12th 2004, when a resounding political issue happened in Korea[1]. It is a typical FE during which a very large amount of clients try to access the server.

The other trace we have is obtained from two trans-pacific T-3 links connecting the United States and a Korean Internet gateway between 9:36am and 9:55am on Dec 14th 2001 (denoted KRUS trace). We extract two FE web logs, 20 minutes each, from this trace. The first one, denoted FE01, contains an FE in which people tried to download newly issued versions of decorating pictures and java scripts for their personal websites and blogs. This is an FE as the newly released pictures and scripts received a lot of attractions and the server received a very large amount of requests during the first few minutes[2]. The second one, denoted FE02, contains requests to a Microsoft Windows update website which attracted a huge number of requests when an accumulated patch to Windows Internet Explorer was released. We analyze the KRUS traces with a 10-second time interval (because the events are relatively short) and examine the MBC trace with a 1-minute time interval (because this event is relatively long).



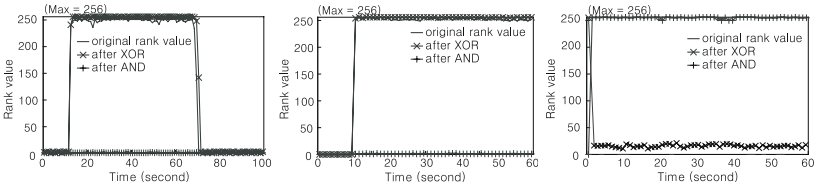
(a) FE01 (b) FE02 (c) MBC

Fig. 2. Randomness check on FE traces

We get the DDoS traces in two ways. One is by applying a well-known DDoS detection technique [8] to process the KRUS trace we have. By applying this DDoS detection technique, we managed to find two traces, namely DDoS01 and DDoS02, in which the source IP addresses are spoofed. We also generated a non-source-spoofed DDoS attack traces with the normal web requests as background traffic using NS-2. We use the CAPBELL/SINGLEBELL topology [1] to simulate a client/server environment in which there are more than 1000 attackers attacking the web server using UDP flooding and 2500 legitimate clients accessing the web server. Note that the CAPBELL/SINGLEBELL topology [1]

[1] During these two days, it was reported that the President of the United States and his top advisers have received intelligence reports describing a confusing series of actions by North Korea that some experts believe could indicate the country is preparing to conduct its first test explosion of a nuclear weapon. (http://www.nytimes.com/2004/09/12/international/asia/12nuke.html).

[2] http://files.cometsystems.com



(a) DDoS01 (b) DDoS02 (c) Non source-spoofed DDoS

Fig. 3. Randomness check on DDoS traces

and the corresponding simulation scenarios are constructed by processing many real ISP traces, which serve as a very good approximation as a real client/server network. The bandwidth between each client and the server varies from 40Mb to 100Mb and the latency between varies from 20ms to 80ms in a uniform distribution. Using the Pareto-law, the 2500 legitimate clients are distributed in 175 clusters.

4.2 Results and Discussion

In this subsection, we show the results of applying our technique FDD to the three FE traces and three DDoS traces described in the previous sections. Fig. 2 and Fig. 3 show the results of three rank values of the matrices: the original rank value, the one after the XOR operation, and the one after the AND operation.

Fig. 2 shows the results for the three FE traces discussed in Section 4.1. We see that although original rank values ($R(M_t)$) are always the larger than the rank values of the matrices after the XOR operation ($R_{\text{XOR}}(M_t)$) and the AND operation ($R_{\text{AND}}(M_t)$), $R(M_t)$ is never above 252 for an extended period of time[3]. This suggests that 252 could be the threshold T in order to accurately classify these events as FEs. (Please refer to Appendix B for details in calculating this threshold.) Also note that the results for both FE01 and FE02 are way below the threshold 252 in our evaluations. As indicated in Section 3.4, the main reason for having medium rank values of $R(M_t)$ is that the distribution of clients among clusters is not random — a large amount of requests come from a small number of clusters. $R_{\text{XOR}}(M_t)$ is smaller because the XOR operation removes cluster overlapping (cluster overlapping is common in FEs [11]), while $R_{\text{AND}}(M_t)$ is small but not negligible because the AND operation removes all but the overlapping clusters, which still consists of a non-negligible amount of requests (during FE, there are some new requests coming every now and then).

Fig. 3 shows the results for the three DDoS traces discussed in Section 4.1. In these cases, we notice that two of the three rand values are way above the threshold 252, which indicates that we will be able to distinguish FE from DDoS

[3] The original rank value $R(M_t)$ for MBC may not be very clear in the graph. It is right above the line for $R_{\text{XOR}}(M_t)$.

by setting the threshold 252. In the case of source-spoofed DDoS (DDoS01 and DDoS02), $R_{\text{XOR}}(M_t)$ exceeds the threshold because source spoofing results in relatively small cluster overlapping, whereas in the case of non-source-spoofed DDoS, cluster overlapping is very big, which makes $R_{\text{AND}}(M_t)$ exceed the threshold.

5 Conclusion

In this paper, we propose a simple yet effective mechanism FDD to distinguish flash event and distributed denial of service attacks using randomness check. With the help of our matrix construction using the incoming IP address, as well as the XOR and AND operations on the matrices, we manage to apply matrix randomness check to distinguish FE and DDoS. Our trace-driven evaluation results show that FDD distinguishes between FE and DDoS attacks with high accuracy and low memory usage.

References

1. Feldman, A., Gilbert, A.D., Huang, P., Willinger, W.: Dynamics of IP traffic: A study of the role variability and the impact of control. In: ACM SIGCOMM (1999)
2. Rukhin, A., Soto, J., Nechvatal, J., Smid, M., Barker, E., Leigh, S., Levenson, M., Vangel, M., Banks, D., Heckert, A., Dray, J., Vo, S.: A statistical test suite for random and pseudorandom number generators for cryptographic applications, May 2001, vol. 800(22). NIST Special Publication (2001)
3. Stavrou, A., Keromytis, A.D.: Countering DoS attacks with stateless multipath overlays. In: ACM Computer and Communication Security (November 2005)
4. Krishnamurthy, B., Wang, J.: On network-aware clustering of web clients. In: ACM SIGCOMM (August 2000)
5. Moore, D., Voelker, G.M., Savage, S.: Inferring internet Denial-of-Service activity. In: USENIX Security Symposium (2001)
6. Marsaglia, G., Tsay, L.H.: Matrices and the structure of random number sequences. Linear Algebra Appl. Elsevier Science 67, 147–156 (1985)
7. Marsaglia, G.: Diehard: A battery of tests of randomness (1996), http://stat.fsu.edu/~geo/diehard.html
8. Kim, H., Bahk, S., Kang, I.: Real-time visualization of network attacks on high-speed links. IEEE Network Magazine 18, 30–39
9. Park, H., Lee, H., Kim, H.: Detecting unknown worms using randomness check. IEICE Trans. Communication E90-B(4), 894–903 (2007)
10. Wang, H., Zhang, D., Shin, K.G.: Detecting SYN flooding attacks. IEEE INFOCOM2002 3, 1530–1539 (2002)
11. Jung, J., Krishnamurthy, B., Rabinovich, M.: Flash crowds and denial of service attacks: Characterization and implications for CDNs and web sites. In: World Wide Web (May 2002)
12. Argyraki, K.: Active internet traffic filtering: real-time response to Denial-of-Service attacks. In: USENIX Annual Technical Conference (April 2005)
13. Adamic, L.A.: Zipf, power-laws, and pareto - a ranking tutorial (1999), http://www.hpl.hp.com/research/idl/papers/ranking/ranking.html

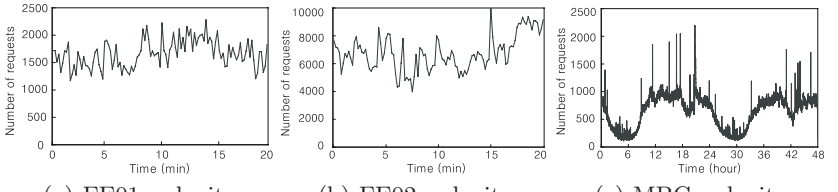
14. Gordon, L.A., Loeb, M.P., Lucyshn, W., Richardson, R.: CSI/FBI computer crime and security survey. In: Computer Security Inst. (2004)
15. Feinstein, L., Schackenberg, D., Balupari, R., Kindred, D.: Statistical approaches to DDoS attack detection and response. In: the DARPA Information Survivability Conference and Exposition(DISCEX 2003) (2003)
16. Niven, L.: Flash crowd, The Flight of the Horse. Ballantine Books (1971)
17. Casado, M., Akella, A., Cao, P., Provos, N., Shenker, S.: Cookies Along trust-boundaries(CAT): accurate and deployable flood protection. In: USENIX Workshop on Steps to Reducing Unwanted Traffic on the Internet(SRUTI) (July 2006)
18. Peng, T., Leckie, C., Rnmamohanarao, K.: Proactively detecting Distributed Denial of Service attacks using source IP address monitoring. In: Networking 2004, pp. 771–782 (2004)
19. He, Y., Chen, W., Xiao, B.: Detecting SYN flooding attacks near innocent side. In: Mobile Ad-hoc and Sensor Network(MSN 2005). LNCS, vol. 3794, pp. 443–452. Springer, Heidelberg (2005)

A Used Traffic Data in Evaluation

In this Appendix, we show that the traffics we used satisfy the characteristics of FE or DDoS [11] in three aspects.

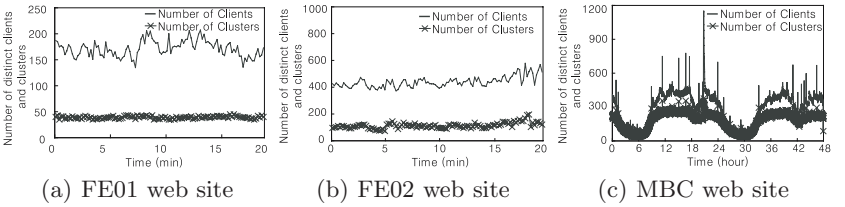
A.1 Traffic During FE

Fig. 4 shows the number of requests connecting to those three web sites within our three traffics(Section 4). Fig. 4 (a) and Fig. 4 (b) show that a large number of requests are coming from clients during 20 minutes. Fig. 4 (c) shows the the number of requests grows dramatically during FE, but the duration of FE in this traffic is relatively short. Fig. 5 shows the numbers of clients and of clusters accessing the web sites. In order to cluster the clients, we employ a network-aware clustering technique [4]. Fig. 6 shows that the distribution of requests among clusters is indeed highly skewed [4] such as Zipf-like distribution. In other words, the distribution of requests among clusters follows Pareto-law, because Zipf, power-law and Pareto can refer to the same thing [13].



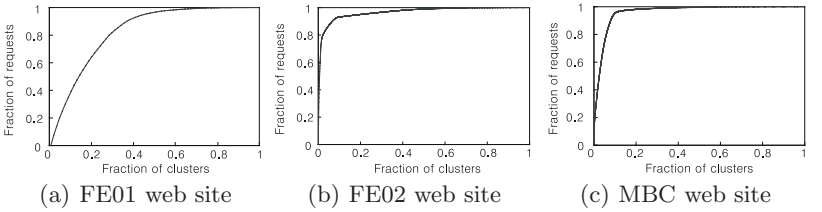
(a) FE01 web site (b) FE02 web site (c) MBC web site

Fig. 4. Traffic volumes



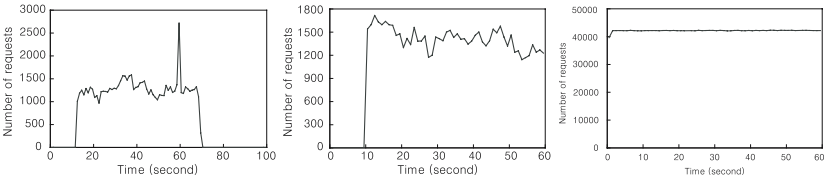
(a) FE01 web site (b) FE02 web site (c) MBC web site

Fig. 5. Distribution of clients and clusters



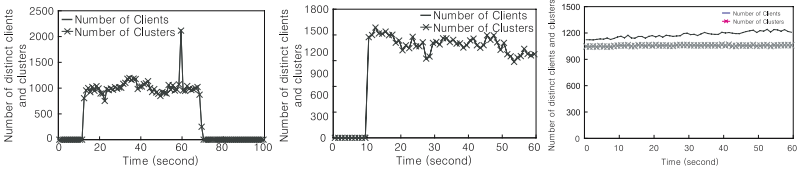
(a) FE01 web site (b) FE02 web site (c) MBC web site

Fig. 6. Cluster contribution to requests



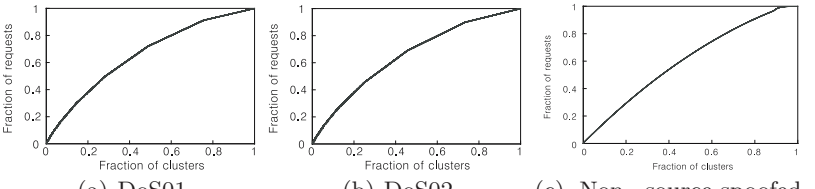
(a) DoS01 (b) DoS02 (c) Non source-spoofed DDoS

Fig. 7. Traffic volumes



(a) DoS01 (b) DoS02 (c) Non source-spoofed DDoS

Fig. 8. Distribution of clients and clusters



(a) DoS01 (b) DoS02 (c) Non source-spoofed DDoS

Fig. 9. Cluster contribution to requests

A.2 Traffics During DDoS Attacks

In Fig. 7(a) and Fig. 7(b), we present the number of requests attacking two servers from KRUS traffic, while Fig. 7(c) shows the number of requests attacking the simulated server in NS-2 with UDP flooding attack. Fig. 8 shows that the number of clients is similar to that of clusters in three traffics. Fig. 8(a) and Fig. 8(b) show the case in source-spoofed DoS and Fig. 8(c) is for non-source-spoofed DDoS. Besides, in these attacks, as we can see in Fig. 9, the distribution of clients among clusters fails to follow Pareto-law.

B Calculation of the Threshold

From Eq. 5, we start calculating the threshold of the rank value of the matrix by letting the equation equal to a value P.

$$2^{r(n+m-r)-nm} \prod_{i=0}^{r-1} \frac{\left(1-2^{i-n}\right)\left(1-2^{i-m}\right)}{\left(1-2^{i-r}\right)} = P \tag{6}$$

where P is the probability of which the matrix will not be random.

$$\log_2 \left(2^{r(n+m-r)-nm} \prod_{i=0}^{r-1} \frac{\left(1-2^{i-n}\right)\left(1-2^{i-m}\right)}{\left(1-2^{i-r}\right)} \right) = \log_2 P \tag{7}$$

Let $m = n$ for our square matrix so that we can get the following equation.

$$2mr - r^2 - m^2 + \log_2 \prod_{i=0}^{r-1} \frac{\left(1-2^{i-m}\right)^2}{\left(1-2^{i-r}\right)} = \log_2 P \tag{8}$$

Through mathematical induction, $\log_2 \prod_{i=0}^{r-1} \frac{\left(1-2^{i-m}\right)^2}{\left(1-2^{i-r}\right)}$ would have the biggest value when r is 1 and the value of m is fixed, and since this biggest value is smaller than 1, we can have the following equation.

$$(m-r)^2 > \log_2 \frac{1}{P} \tag{9}$$

Following the above procedure, if we assume P is 0.01% (a value near to zero), we will get 252 as the biggest value of r to be the threshold, when the value of m is 256.

Antisocial Networks: Turning a Social Network into a Botnet

E. Athanasopoulos[1], A. Makridakis[1], S. Antonatos[1], D. Antoniades[1], S. Ioannidis[1], K.G. Anagnostakis[2], and E.P. Markatos[1]

[1] Institute of Computer Science (ICS)
Foundation for Research & Technology Hellas (FORTH)
{elathan,amakrid,antonat,danton,sotiris,markatos}@ics.forth.gr
[2] Institute for Infocomm Research, Singapore
kostas@i2r.a-star.edu.sg

Abstract. *Antisocial Networks* are distributed systems based on social networking Web sites that can be exploited by attackers, and directed to carry out network attacks. Malicious users are able to take control of the visitors of social sites by remotely manipulating their browsers through legitimate Web control functionality such as image-loading HTML tags, JavaScript instructions, *etc.* In this paper we experimentally show that Social Network web sites have the ideal properties to become attack platforms.

We start by identifying all the properties of Facebook, a real-world Social Network, and then study how we can utilize these properties and transform it into an attack platform against any host connected to the Internet. Towards this end, we developed a real-world Facebook application that can perform malicious actions covertly. We experimentally measured it's impact by studying how innocent Facebook users can be manipulated into carrying out a Denial-of-Service attack. Finally, we explored other possible misuses of Facebook and how they can be applied to other online Social Network web sites.

1 Introduction

The massive adoption of social networks by Internet users provides us with a unique opportunity to study possible exploits that will turn them into platforms for antisocial and illegal activities, like DDoS attacks, malware propagation, spamming, privacy violations, *etc.* We define *antisocial networks* as a *social network, deviously manipulated for launching activities connected with fraud and cyber-crime.*

Social networks have by nature some intrinsic properties that make them ideal to be exploited by an adversary. The most important of these properties are: (*i*) a very large and highly distributed user-base, (*ii*) clusters of users sharing the same social interests, developing trust with each other, and seeking access to the same resources, and (*iii*) platform openness for deploying fraud resources and applications that lure users to install them. All these characteristics give

T.-C. Wu et al. (Eds.): ISC 2008, LNCS 5222, pp. 146–160, 2008.

adversaries the opportunity to manipulate massive crowds of Internet users and force them to commit antisocial acts against the rest of the Internet, without their knowledge. In this paper we explore these properties, develop a real exploit, and analyze its impact.

The main contributions of this paper is a first investigation into the potential misuse of a social network for launching DDoS attacks on third parties. We have built an actual Facebook application, that can turn its users into a FaceBot. We used our FaceBot to carry out a complete evaluation of our proof-of-concept attack via real-world experiments. Extrapolating from these measurements along with popularity metrics of current Facebook applications, we show that owners of popular Facebook applications have a highly distributed platform with significant attack firepower under their control.

2 Related Work

The structure and evolution of social networks has been extensively studied [18,9,11], but little work has been done on measuring real attacks on these sites. The most closely related work to our paper was done by Lam *et al.* in [17]. Our work here extends the idea of Puppetnets by taking into account the characteristics of a special kind of Internet systems which rely heavily on the social factor: social network web sites. The authors of [17] omit explaining *how* they will make their Web site popular, in order to carry out the attack. We on the other hand are taking advantage of already popular Web sites like `facebook.com`. Such sites prove to be ideal for carrying Puppetnet type attacks.

Jagatic *et al.* in [16] study how phishing attacks [13] can be made more powerful by extracting information from social networks. Identifying groups of people leads to more successful phishing attacks than by simply massively sending e-mails to random people unrelated to each other. However, apart from scattered blog entries that report isolated attacks (such as malware hosting in Myspace [4]), there have been no large-scale attacks to social networks, or using social networking sites, reported or studied so far.

In the space of peer-to-peer systems, there have been a few attacks that have appeared and have been analyzed by researchers. One may consider a peer-to-peer system to be similar to a social network in the sense that there are millions of users that connect to each other forming a network. Gnutella, an unstructured peer-to-peer file sharing system, has been used in the past as an attack platform [10]. In a similar fashion, the work in [19,21] presented how Overnet and KAD can be misused for launching Denial of Service attacks to third parties. Finally, in [12], the authors have managed to transform BitTorrent to a platform for similar attacks.

3 Background

Social Networks. Social networking sites are becoming more popular by the day. Millions of people daily use social networking sites such as `facebook.com`,

`LinkedIn.com`, `Myspace.com` and `Orkut.com`. Some of them are used for professional contacts, *e.g.* LinkedIn, while others are primarily used for communication and entertainment. The structure of a social networking site is quite simple. Users register to the site, create their profile describing their interests and putting some personal information, and finally add friends/contacts to their profile. Adding a friend involves a confirmation step from the other party most of the times. The view of a user's profile is usually limited to the friends of that user, unless the user wants the profile to be public. In that case, all users of the site can view it. Social networking sites also support the creation of groups and networks.

Facebook is considered to be one of the most popular social networking sites. It started as a project of a student to keep track of schoolmates but has now grown up to serve more than 64 million people from around the world, with an average of 250,000 new registrations per day [2]. Facebook has a very interesting feature, the Facebook applications. Facebook builders have implemented a platform on top of which developers can build complete applications. In the *Facebook Platform* any developer with a good idea and basic programming skills can create one. Over 200,000 developers have done so, as reported by Adonomics [1]. Users can add these applications to their profile and invite their friends to add them too. A constraint put by Facebook is that invitations are limited to up to 20 friends per day. Typical applications involve solving a quiz, filling questionnaires, playing games and many more. Up to date, the number of Facebook applications has surpassed fifteen thousand. Facebook applications can be considered as XHTML snippets that inherit all properties of web applications.

Puppetnets. Puppetnets [17] exploit the design principles of World Wide Web. Web pages can include links to elements located at different domains, other than the one they are hosted at. A malicious user can craft special pages that contain thousands of links pointing at a victim site. When an unsuspecting user visits that page, her browser starts downloading elements from the victim site and thus consuming its bandwidth. The firepower of this attack increases with the popularity of the malicious page, similar to the slashdot effect [15].

Puppetnets use a number of techniques to make the attacks more effective. The use of JavaScript permits more flexible and powerful attacks as unsuspecting users can repeatedly download elements from victim sites or perform other kinds of attacks, such as port scanning and computational attacks. The firepower of Puppetnets depends on three main factors. First, the popularity of the malicious page. Second, the duration of visits to the malicious page. The more the unsuspecting user stays on the malicious page, the longer the attack takes place in the background. Third, the bandwidth of unsuspecting users and their latency to the victim site. These factors determine the number of downloads per second an attacker can achieve.

4 Experimental Evaluation

In this section we experimentally evaluate the firepower of a FaceBot. Specifically, we explore the effect of placing a malicious Facebook application, which exports HTTP requests to a victim host. We have conducted experiments, using

a *least effort* approach. By using the term of *least effort* we mean that during the whole study we did the *least* we could do in terms of spending resources, adding complexity and enhancing our developments with obscure and hackish features, which could lead in overestimated results. For example, during the deployment of a Facebook application we *did not add special obligatory massive invitation features* for boosting the application's propagation in the social network. In section 5, based on our experimental results, we extrapolate the firepower of FaceBot, by examining the popularity of existing Facebook applications.

4.1 Experimental Setup

Our initial vision is to create a first *proof-of-concept* FaceBot for demonstration purposes, while at the same time not causing any harm to real Facebook users. Furthermore, our experiment was conducted using the *real* social network website, namely `facebook.com`.

We created a real-world Facebook application, which we call *Photo of the Day* [8], that presents a different photo from National Geographic to facebook users every day. In order to keep the experiment in a *least effort* approach, we didn't employ any obligatory invitations during its installation in a user's profile.[1] However, we did announce the application to members of our research group and we encouraged them to propagate the application to their colleagues. To our surprise, the application was installed by a significant Facebook population, which was completely unaware to us (see our popularity results, later in this section).

Every time a user clicks on the *Photo of the Day* application, an image from the respective service of National Geographic[2] appears [7]. However, we have placed special code in the application's source code, so that every time a user views the photo, HTTP requests are generated towards a victim host. More precisely, the application embeds four hidden frames with inline images hosted at the victim. Each time the user clicks inside the application, the inline images are fetched from the victim, causing the victim to serve a request of 600 KBytes, but the user is not aware of that fact (the images are never displayed). We list a portion of our sample source code which is responsible for fetching an inline image from a victim host and placing it to a hidden frame inside the *Photo of the Day* application, in Figure 1.

For our experiments, the victim Web server which hosts the inline images is located in our lab, isolated from any other network activity. In the following subsection we present the results associated with the traffic experienced by our Web server.

[1] It is very common that Facebook applications require a user to invite a subset of her friends, and thus advertize the application to the Facebook community, prior the installation. This practice helps in the further propagation of the application in Facebook. Typically, a user must announce the application to about 20 of her friends in order to proceed with the installation.

[2] National Geographic has specific terms for content distribution, which are not violated by this work [6].

```
<iframe name="1" style="border: 0px none #ffffff;
width: 0px; height: 0px;"
src="http://victim-host/image1.jpg?
fb_sig_in_iframe=1&amp;
fb_sig_time=1202207816.5644&amp;
fb_sig_added=1&amp;
fb_sig_user=724370938&amp;
fb_sig_profile_update_time=1199641675&amp;
fb_sig_session_key=520dabc760f374248b&amp;
fb_sig_expires=0&amp;
fb_sig_api_key=488b6da516f28bab8a5ecc558b484cd1&amp;
fb_sig=a45628e9ad73c1212aab31eed9db500a">
</iframe><br/>
```

Fig. 1. Sample code of a hidden frame, inside a Facebook application, which causes an image, namely `image1.jpg` to be fetched from `victim-host`

4.2 Attack Magnitude

In Figure 2 we present the number of requests per hour recorded by our Web server from the time the *Photo of the Day* application was uploaded to `facebook.com` and for a period of a few days. Notice, that the request rate reached a peak of more than 300 requests/hour after a few days from the publication time. During the peak day of January 29th, our Web server recorded an excess of 6 Mbit per second of traffic (see Figure 3). The request rate shown in Figure 2, as well as the outgoing traffic shown in Figure 3, is purely Facebook related. We can isolate the packets originating from users accessing `facebook.com` by inspecting the *referer* field[3]. We further discuss the importance of the referer field in Section 6.

It is important to note that the request rate per hour never fell below a few tens of request and during peak times it reached a few hundred of requests. Notice, that depending on the nature of the malicious Facebook application, the request rate may differ substantially. In our experiment, each user was generating only four requests towards our Web server per application visit. We further explore the nature of a malicious Facebook application in Section 5.

It is also interesting to notice that the traffic pattern is quite bursty (see Figure 3). This is related to the *social nature* of the attack platform. Users seem to visit Facebook also in bursty fashion (approximately at the same time). This is more clearly presented in Figure 4, where we plot the distribution of user inter-arrival times (the times at which users visit the Photo of the Day application) for the 29th of January. We calculated this distribution using the entry points to the Photo of the Day application as they were recorded by our victim Web server. The users' inter-arrival distribution indicates that a typical inter-arrival time has a period from a few tens of seconds to a few minutes. Note, that during the 29th of January, according to Figure 8, our proof of concept application recorded 480 Facebook daily active users.

To further verify our feelings about the bursty nature of the traffic we were experiencing in the victim host, we installed two sensors and captured traffic emitted by Facebook users. The first sensor was installed in an academic institute

[3] `http://www.w3.org/Protocols/HTTP/HTRQ_Headers.html#z14`

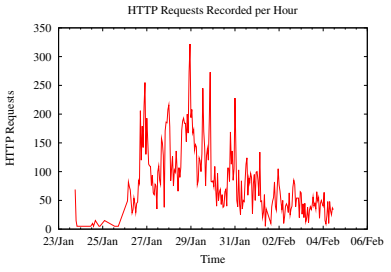


Fig. 2. The HTTP requests as were recorded by the victim Web server

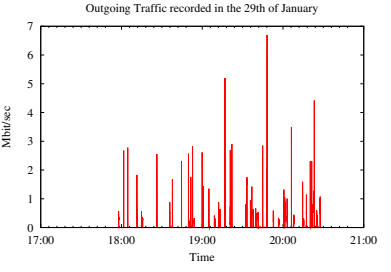


Fig. 3. Bandwidth use at the victim Web server during the attack on 29/01/2008

and was able to monitor approximately 120,000 IP addresses. We recorded 100 unique Facebook users in a monitoring period of 1 day. The second sensor was installed in a /16 enterprise network. We recorded 75 unique Facebook users in a monitoring period of 5 days. We used the collected traces from these sensors in order to calculate the user requests' inter-arrival distribution at Facebook. We present the results in Figure 5. It is evident that small inter-arrival periods characterize the requests made by Facebook users. Note, that users arrive in bursts to their home pages in facebook.com, but this does not immediately imply that they will use the Photo of the Day application.

To summarize, based on the spontaneous peaks in Figures 2 and 3, and considering the fact that Facebook users are arriving nearly at the same time (see Figure 4), we conclude that a malicious Facebook application can absorb Facebook users and force them to generate HTTP requests to a victim host in burst mode fashion.

Notice, that our malicious application was absorbing a fixed amount of traffic from the victim host. An adversary could employ more sophisticated techniques and create a JavaScript snippet, which continuously requests documents from a victim host over time. In this way the attack may be significantly amplified. In Figure 6 we plot typical session times of Facebook users, as were recorded by our two sensors. Observe that a typical user session of a Facebook user ranges from a few to tens of minutes.

4.3 Attack Distribution

Using the IP addresses recorded in the logs of our victim Web server, we tried to identify the geographical origin of each Facebook user. Our main interest was to investigate how distributed can an attack based on a social web site, like `facebook.com`, be. We used the `geoip` tool[3], in order to map our collected IPs to actual countries. We ignored the fact that some Facebook users might be using some sort of an anonymizing system like TOR [14], because our goal was not to capture the *origin of the users*, but the *origin of the requests*, which were recorded by our victim host.

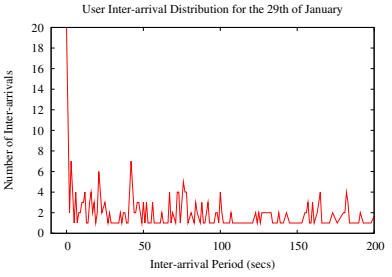


Fig. 4. The distribution of user inter-arrival times at the victim site on 29/01/2008, with over 480 users recorded as active

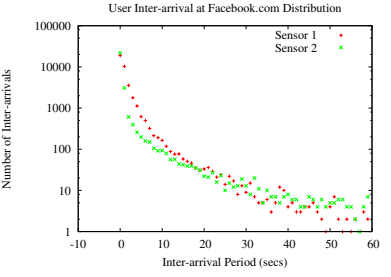


Fig. 5. The distribution of user inter-arrival periods at `facebook.com` for one day. Our two sensors recorded 100 and 75 unique users respectively.

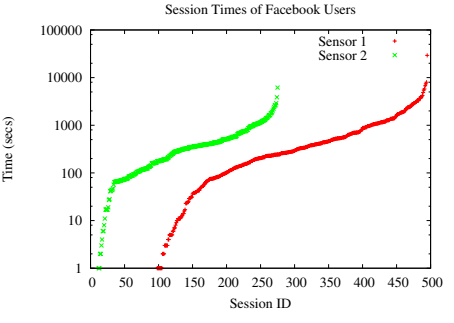


Fig. 6. Session times of Facebook users as were recorded by our two sensors. The first sensor recorded 495 user sessions and the other one recorded 275 user sessions.

In Figure 7 we are marking in black every country from which we recorded at least one request. It is evident that the nature of a FaceBot, even one that is a proof of concept, is highly distributed.

4.4 Tracking Popularity

In Figure 8 we explore the popularity of our proof of concept Facebook application, as it is measured by Adonomics [1]. Recall that, as we stated multiple times in this section, we followed a *least effort* approach, which means that we did not employed sophisticated methods for advertizing our application to `facebook.com`. However, as it is evident from Figure 8, our application was installed by nearly 1,000 different users in the first few days. This is rather impressive correlating it with statistics related to commodity software downloads. For example, it took months for the most successful project in SourceForge.com to reach thousands of downloads[4].

[4] eMule Statistics: `http://sourceforge.net/project/stats/?group_id=53489&ugn=emule&type=&mode=alltime`

Fig. 7. Location of FaceBot hosts. Countries coloured in black hosted at least one FaceBot participant.

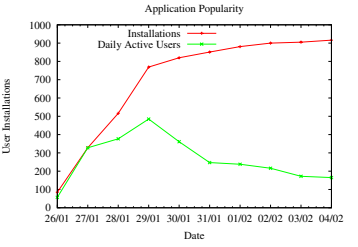


Fig. 8. The popularity of the *Photo of the Day* application, as it is tracked by Adonomics.com

5 Attack Firepower

Based on the experimental results from the previous section we proceed to estimate the firepower of a large FaceBot. For this we are going to assume that an adversary has developed a *highly popular* Facebook application, which employs the tricks we presented in the previous sections.

We denote with $F(t)$ the distribution of outgoing traffic a victim Web server exports, due to Facebook requests, over time. This is essentially the firepower of a FaceBot. In section 4 we experimentally measured this distribution for a proof of concept FaceBot and we presented our results in Figure 3. Our aim, in this section, is to find an analytical expression for $F(t)$.

We denote with a_{out} the outgoing traffic a Facebook application can pull from a victim host, once the user on that host is tricked into using the malicious application. Even though sophisticated use of client side technologies (like JavaScript) can make a_{out} a function over time (*e.g.*, a malicious JavaScript snippet can generate requests towards a victim host in an infinite loop), for simplicity we assume that a_{out} is a fixed quantity.

We denote with $U(t)$ the number of users accessing this application over time. It follows that:

$$F(t) = a_{out}U(t) \tag{1}$$

To estimate $U(t)$, we need the following: (a) the number of active users over a period P, and (b) an estimation of the users' inter-arrival times. If we denote the active users with $u(t)$ and the inter-arrival distribution with $u_r(t)$, then:

$$U(t) = \frac{\int_0^P u(t)dt}{u_r(t)} \tag{2}$$

Assuming that there is a FaceBot based on a highly popular Facebook application and that we want to estimate its firepower at time T, F_T, we can use the average of the inter-arrival distribution, and thus:

$$F_T = a_{out} \frac{\int_0^P u(t)dt}{< u_r >} \quad (3)$$

For example, if we have a FaceBot with $a_{out} = 10Kbit/sec$, which is installed by 1,000 users, from whom 100 were active in the period of 10 seconds and their average inter-arrival time was 2 secs, then $F_{(10)} = 10Kbit/sec\frac{100}{2} = 0.5Mbit/sec$.

In Table 1 we list the Top-5 Facebook applications as of early February 2008, according to Adonomics.com[1]. These applications have from 1 million to more than 2 millions of daily active users. The user-base of these applications is so large, that we can assume that the user inter-arrival time follows a uniform distribution.[5] We further assume that an adversary has deployed one of these applications, which has 2 million of daily active users. That is, assuming uniform user inter-arrival time, approximately 23 users/sec are using the application. If the adversary has deployed the malicious application with $a_{out} = 1Mbit/sec$[6], then the victim will have to cope with unsolicited traffic of 23 Mbit/sec and during the period of one day will have received nearly 248 GB of unwanted data.

Table 1. The Top-5 of Facebook applications as of the beginning of February 2008, in terms of active users. Source: Adonomics.com[1].

Application	Installations	Daily Active Users
FunWall	23,797,800	2,379,780
Top Friends	24,955,200	2,245,970
Super Wall	23,274,800	1,861,980
Movies	15,934,700	1,274,780
Bumper Sticker	7,989,700	1,118,560

6 Discussion and Countermeasures

From our analysis in Section 5 we can see that an adversary can take full advantage of popular social utilities, to emit a high amounts of traffic towards a victim host. However, apart from launching a DDoS attack to third parties, there are other possible misuses in the fashion of Puppetnets [17]:

– *Host Scanning:* Using JavaScript, an attacker can make an application that identifies whether a host has arbitrary ports open. As browsers impose only few restriction on destination ports (some browsers like Safari even allow connection to sensitive ports like 25), an attacker can randomly select a host and a port, and request an object through normal HTTP requests. Based on the response time, which can be measured through Javascript, the attacker can figure if the port is alive or not.

[5] Having a non-uniform inter-arrival time distribution would further amplify the attack, because the victim host would have to cope with large flash crowd events [15] in very short periods.

[6] The adversary needs to download a file of size of 125 KBytes from the victim, in order to achieve such an a_{out} value.

– *Malware Propagation:* An unsuspecting user can participate in malware and attack propagation. If a server can be exploited by a URL-embedded attack vector, then malicious facebook applications can contain this exploit. Every user that interacts with the application will propagate the attack vector.
– *Attacking Cookie-based Mechanisms:* Similarly to XSS worms, a malicious application can override authentication mechanisms that are based on cookies. Badly-designed sites that support automated login using cookies suffer from such attacks.

Finally, there are other possible misuses of `facebook.com` itself. For example, an adversary can collect sensitive information of `facebook.com` users, without their permission. Facebook.com gives users the opportunity to have their profile locked and visible only by their contacts. However, a `facebook.com` application has full access in all user's details. An adversary could deploy an application, which simply posts all user details to an external colluding Web server. In this way, the adversary can gain access to the personal information of users, who have installed the malicious application.[7]

In the rest of this section we propose countermeasures for defending and preventing a FaceBot based attack.

6.1 Defending Against a FaceBot

To defend against a FaceBot, a victim host must filter out all incoming traffic introduced by Facebook users. Using the referer field of the HTTP requests the victim can determine whether a request originates from `facebook.com` or not, and stop the attack traffic (*e.g.* by using a NIDS or Firewall system). However, it is possible for a Facebook application developer to overcome this situation. With respect to our proof of concept application, which embeds hidden frames with inline images, the strategy would be to create a separate page to load them from. For example the source of the inline frame can be:

```
src="http://attack-host/dummy-page?ref=victim-host/image1.jpg"
```

In this example the *attack host* is the Web server where the source code of the *Photo of the Day* lives. The dummy-page PHP file contains the following code:

```
<?php
if ($_GET["ref"]) { $ref=$_GET["ref"]; }
print("<meta http-equiv='refresh'
content='0; url=$ref'>");
?>
```

By employing this technique, HTTP requests received by the victim host have an empty referer field, giving the attacker a way to hide her identity. This is a typical usage of a reflector [20] by the adversary. Notice however, that the adversary must tunnel the requests to the victim. This means, that the adversary will also receive all the requests targeting the victim, but she will not have to

[7] Indeed, this proved to be possible, while this paper was under the review process [5].

actually serve the requests. Practically, the adversary will receive plain HTTP requests (a few bytes of size each), will have to process them in order to trim the referer related data and then pass it to the victim. On the other hand, the victim will have to serve the requests, which, depending on the files the victim serves, might reach the size of MBytes of information for each server request.

6.2 Preventing a FaceBot

Providers of social networks should be careful when designing their platform and APIs in order to have low interactions between the social utilities they operate and the rest of the Internet. More precisely, social network providers should be careful with the use of client side technologies, like JavaScript, *etc.* A social network operator should provide developers with a strict API, which is capable of giving access to resources only related to the system. Also, every application should run in an isolated environment imposing constraints to prevent the application from interacting with other Internet hosts, which are not participants of the social network. Finally, operators of social networks should invest resources in verifying the applications they host. Regarding our application, the Facebook Platform can cancel the use of `fb:iframe` tag, as this tag is used to load images hosted at the victim host. Currently, developers can not use `fb:iframe` tag on the profile page of a user. [8] Otherwise, the `fb:iframe` tag can be handled in a special manner, as in the case of the `img` tag. When publishing a page, Facebook servers request any image URL and then serve these images, rewriting the `src` attribute of all `img` tags using a *.facebook.com domain. This protects the privacy of Facebook's users and not allow malicious applications to extract information from image requests made directly from a the view of a user's browser. Thus, if the `src` attribute of an `iframe` is an image file (*e.g.* .jpg, .png, *etc.*), the Facebook Platform can handle these frames in a way similar to `img` tags.

7 Conclusion

In this paper we presented *Antisocial Networks* or how it is possible to turn a social network into a botnet that can be used to carry out a number of attacks. We developed FaceBot, an application that can run on `facebook.com`, and carry out DDoS attacks against any host on the internet. Our analysis involved building a real-world facebook.com application, conducting an actual attack on our lab servers, and doing an estimation of the firepower of a FaceBot.

We have shown that applications that live inside a social network can easily and very quickly attract a large user-base (in the order of millions of users) that can be redirected to attack a victim host. We experimentally determined the user-base to be highly distributed, and of a world-wide scale. Finally, we have shown that the victim of a FaceBot attack may be subject to an attack that will cause it to serve data of the magnitude of GigaBytes per day.

[8] `http://wiki.developers.facebook.com/index.php/Fb:iframe`

Acknowledgments

This work was supported in part by the project CyberScope, funded by the Greek Secretariat for Research and Technology under contract number PENED 03ED440. The work was, also, supported by the Marie Curie Actions - Reintegration Grants project PASS. We thank the anonymous reviewers for their valuable comments. Elias Athanasopoulos, Andreas Makridakis, Sotiris Ioannidis, Spiros Antonatos, Demetres Antoniades and Evangelos P. Markatos are also with the University of Crete. Elias Athanasopoulos is also funded from the PhD Scholarship Program of Microsoft Research Cambridge.

References

1. Facebook Analytics and Advertising, http://adonomics.com
2. Facebook Statistics, http://www.facebook.com/press/info.php?statistics
3. Geo IP Tool, http://www.geoiptool.com
4. Hackers crash the Social Networking Party, http://www.pcworld.com/article/id,127347-page,1-c,internettips/article.html
5. Identity 'at risk' on Facebook, http://news.bbc.co.uk/2/hi/programmes/click_online/7375772.stm
6. National Geographic Content Usage, http://www.nationalgeographic.com/community/terms.html#content
7. National Geographic Photo of the Day Utility, http://photography.nationalgeographic.com/photography/photo-of-the-day
8. Photo of the Day, http://www.facebook.com/apps/application.php?id=8752912084
9. Ahn, Y.-Y., Han, S., Kwak, H., Moon, S., Jeong, H.: Analysis of Topological Characteristics of Huge Online Social Networking Sites. In: Proceedings of the 16th International Conference on World Wide Web, (May 2007)
10. Athanasopoulos, E., Anagnostakis, K.G., Markatos, E.P.: Misusing Unstructured P2P Systems to Perform DoS Attacks: The Network That Never Forgets. In: Zhou, J., Yung, M., Bao, F. (eds.) ACNS 2006. LNCS, vol. 3989, pp. 130–145. Springer, Heidelberg (2006)
11. Backstrom, L., Huttenlocher, D., Kleinberg, J., Lan, X.: Group Formation in Large Social Networks: Membership, Growth, and Evolution. In: Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2006), (August 2006)
12. Defrawy, K.E., Gjoka, M., Markopoulou, A.: Bottorrent: Misusing bittorrent to launch ddos attacks. In: Proceedings of the USENIX 3rd Workshop on Steps Towards Reducing Unwanted Traffic on the Internet (SRUTI) (2007)
13. Dhamija, R., Tygar, J.D., Hearst, M.: Why phishing works. In: CHI 2006: Proceedings of the SIGCHI conference on Human Factors in com puting systems, pp. 581–590. ACM Press, New York (2006)
14. Dingledine, R., Mathewson, N., Syverson, P.: Tor: The Second-Generation Onion Router. In: Proceedings of the 13th USENIX Security Symposium (August 2004)
15. Halavais, A.: The Slashdot Effect: Analysis of a Large-Scale Public Conversation on the World Wide Web (2001)
16. Jagatic, T.N., Johnson, N.A., Jakobsson, M., Menczer, F.: Social phishing. Commun. ACM 50(10), 94–100 (2007)

17. Lam, V.T., Antonatos, S., Akritidis, P., Anagnostakis, K.G.: Puppetnets: misusing web browsers as a distributed attack infrastructure. In: CCS 2006: Proceedings of the 13th ACM conference on Computer and communications security, pp. 221–234. ACM, New York (2006)
18. Mislove, A., Marcon, M., Gummadi, K.P., Drushcel, P., Bhattacharjee, B.: Measurement and Analysis of Online Social Networks. In: Proceedings of the Internet Measurements Conference (IMC 2007) (2007)
19. Naoumov, N., Ross, K.: Exploiting P2P systems for DDoS attacks. In: InfoScale 2006: Proceedings of the 1st international conference on Scalable information systems, p. 47. ACM Press, New York (2006)
20. Paxson, V.: An analysis of using reflectors for distributed denial-of-service attacks. SIGCOMM Comput. Commun. Rev. 31(3), 38–47 (2001)
21. Steiner, M., Biersack, E.W., En-Najjary, T.: Exploiting kad: Possible uses and misuses. Computer Communication Review 37(5) (2007)

Appendix

Facebook Architecture

Facebook provides all the essentials needed for easy deployment of applications that live inside the social network itself. A user who wants to build a Facebook application must simply add the *Developer Application*[9] to her account. The server side part of the application can be developed in PHP or Java. One major requirement is the presence of a Web server for hosting the new application. Using the Developer Application the developer fills out a form and submits the application. The form has fields, such as the application's name, the IP address of the Web server, *etc.* Typically, after a few days the Facebook Platform Team notifies the developer either that the application was successfully accepted or that it was rejected. Facebook Platform provides the Facebook Markup Language[10] (FBML), which is a subset of HTML along with some additional tags specific to Facebook. Also, the Facebook Query Language[11] (FQL) allows the developer to use an SQL-style interface to easily query some Facebook social data, such as the name or profile picture of a user. Finally, Facebook JavaScript[12] (FBJS) permits developers to use it in their applications. All the above tools give an open API to the developer for easy creation of Web applications that live inside Facebook and which are freely available to every Facebook user.

From Facebook to FaceBot. To exploit a social site, like Facebook, for launching DoS attacks, the adversary needs to create a malicious application, which embeds URIs to a victim Web server. These URIs must point to documents hosted by the victim, like images, text files, HTML pages, *etc.* When a user interacts with the application, the victim host will receive unsolicited requests. These requests are triggered through Facebook, since the application lives inside the social network, but they are actually generated by the Web browsers used by the users that access the malicious application. We define as *FaceBot* the collection of the users' Web browsers that are forced to generate requests upon viewing a malicious Facebook application. Schematically, a FaceBot is presented in Figure 9. The cloud groups a collection of Facebook users who browse a malicious application in Facebook. This causes a series of requests to be generated and directed towards the victim.

One crucial thing to note is that the application is hosted by the developer. That means that if an adversary wants to develop a malicious application, they must also host it. In other words, the adversary has to be able to cope with requests from users that are accessing the application. However, this can be overcomed using a free hosting service, specifically designed for Facebook applications.[13] But even if such a service were not available, the adversary has to cope

[9] http://www.facebook.com/developers/
[10] http://wiki.developers.facebook.com/index.php/FBML
[11] http://wiki.developers.facebook.com/index.php/FQL
[12] http://wiki.developers.facebook.com/index.php/FBJS
[13] Joyent Free Accelerator: http://joyent.com/developers/facebook/

with much less traffic than the one that targets the victim. We further discuss this issue in Section 5.

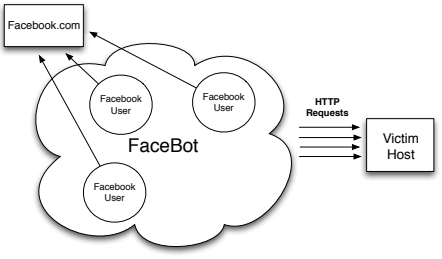


Fig. 9. The architecture of a FaceBot. Users access a malicious application in the social site (`facebook.com`) and subsequently a series of HTTP requests are created, which target the victim host.

Compromising Anonymity Using Packet Spinning

Vasilis Pappas, Elias Athanasopoulos, Sotiris Ioannidis, and Evangelos P. Markatos

Institute of Computer Science (ICS)
Foundation for Research & Technology Hellas (FORTH)
{vpappas,elathan,sotiris,markatos}@ics.forth.gr

Abstract. We present a novel attack targeting anonymizing systems. The attack involves placing a malicious relay node inside an anonymizing system and keeping legitimate nodes *"busy."* We achieve this by creating circular circuits and injecting fraudulent packets, crafted in a way that will make them spin an arbitrary number of times inside our artificial loops. At the same time we inject a small number of malicious nodes that we control into the anonymizing system. By keeping a significant part of the anonymizing system busy spinning useless packets, we increase the probability of having our nodes selected in the creation of legitimate circuits, since we have more free capacity to route requests than the legitimate nodes. This technique may lead to the compromise of the anonymity of people using the system.

To evaluate our novel attack, we used a real-world anonymizing system, TOR. We show that an anonymizing system that is composed of a series of relay nodes which perform cryptographic operations is vulnerable to our packet spinning attack. Our evaluation focuses on determining the cost we can introduce to the legitimate nodes by injecting the fraudulent packets, and the time required for a malicious client to create *n-length* TOR circuits. Furthermore we prove that routers that are involved in packet spinning do not have the capacity to process requests for the creation of new circuits and thus users are forced to select our malicious nodes for routing their data streams.

1 Introduction

Anonymizing systems have been steadily becoming popular as network users that want to hide their identity are using them when accessing Internet services, like Web browsing and Instant Messaging. Anonymity in networks dates back to more than twenty years, when Chaum [7] introduced the concept of anonymous communications. Over the last ten years there have been a series of proposals for anonymizing systems for numerous services. We refer the reader to [8, 10, 12, 16, 18, 19, 20, 22] for some of the most popular anonymizing system proposals.

Following user needs, anonymizing systems have moved from being purely academic proposals and have been deployed as real-world infrastructures. One of the most popular existing solutions for using the Internet in an anonymous

T.-C. Wu et al. (Eds.): ISC 2008, LNCS 5222, pp. 161–174, 2008.

fashion is the TOR system [11]. TOR has been specifically designed for providing anonymity for low-latency services such as the World Wide Web.

TOR is composed of a collection of routing nodes that are available for circuit creations between entities that want to communication in an anonymous fashion. For example, if Alice wants to surf on Bob's web site using TOR, she will have to pick three or more available TOR routers, create a circuit that end's at Bob's web site, and proceed to tunnel her requests over that circuit. Bob on his end will never come in direct contact with Alice but only with a TOR router. This way Alice's anonymity is preserved.

Theoretically, it is possible for an attacker to place malicious nodes inside an anonymity network that is circuit-based (like TOR) and manage to compromise Alice's anonymity, should those malicious nodes are selected in Alice's circuits. However, when such systems contain thousands of routers, the probability of being selected is relatively low, unless the attacker injects large numbers of malicious nodes. In this paper we present a novel attack in which a malicious user injects just a few nodes in system like the above in order to keep legitimate routers busy. At a later stage, the attacker can add malicious nodes that will be relatively idle, and in this way increasing the probability of having them selected.

The rest of this paper is organized as follows. We discuss and compare to prior work in Section 2. In Section 3 we give a detailed presentation of the basic concept of the packet spinning attack. The evaluations of the attack and its magnitude is presents in Section 4. Based on our experimental findings we show how an attacker can actually compromise anonymity in TOR in Section 5. In Section 6 we propose *Tree Based Circuits*, a countermeasure aimed at defeating packet spinning attacks. Finally we conclude and discuss future work in Section 7.

2 Related Work

There are numerous research papers identifying possible attacks against modern anonymizing systems. One fundamental attack against anonymizing systems is based on *traffic analysis* (see, e.g., [3, 17]). Traffic analysis, in the context of anonymizing, is the process of passively monitoring streams of an anonymizing system and trying to correlate them by identifying specific patterns, aiming to reveal the sender or the recipient of an anonymous communication. Traffic analysis has evolved [9, 14] to a practical way of breaking the anonymity provided by an anonymizing system. For example, Danezis *et al.* have shown how traffic analysis can successfully break the anonymity provided by TOR [15].

Apart from traffic analysis, there are other possible attacks against anonymizing systems, and our work here is more closely related to those, since we don't use any traffic analysis to carry out our attack. More precisely, previous work has presented a study on attacks against anonymizing systems which are based in open MIX routers [5]. With respect to TOR, and not considering traffic analysis, there are two major attacks on its anonymizing scheme. The first one suggests to inject malicious nodes in a TOR overlay that lie about their available bandwidth and consequently are selected for TOR circuits with higher probability [4]. In the

second one, Danezis *et al.* are taking advantage of the circuit creation process to compromise TOR [6]. Specifically, when a malicious node realizes that it is unable to compromise a TOR circuit, meaning it is not an entry or an exit node in a TOR circuit, it breaks the circuit and it forces the user to initiate a new one. It uses this technique repeatedly with the hope that the new circuit will contain the malicious node as an entry or exit node.

Borisov *et al.* recentlyll proposed an opportunistic bandwidth measurement algorithm for TOR to replace self reported values [21]. This technique addresses attacks like the one described above [4] and also has a good impact on TOR's overall performance because it achieves better load balancing. Surprisingly, this technique is beneficial for our attack. In the current implementation of TOR the ORs advertise the same bandwidth both under normal conditions and under the LOOP phase (where our spinning cells consume most of their bandwidth). Using the Borisov's technique though, after the LOOP phase our idle malicious ORs will be more likely to be chosen, because the legitimate ones will advertise less available bandwidth.

3 Packet Spinning Attack

In this section we describe in details the packet spinning attack. First, we present the basic idea and then we focus on the parameters which are critical and can make the attack stronger. Even though the attack is feasible in any anonymizing system which uses intermediate relay nodes for hiding the identity of a packet sender, and each relay node is involved in some cryptographic operations, in this analysis we will focus on the TOR anonymizing system.

Overview. The TOR anonymizing system is composed of a collection of nodes[1] that relay traffic from a user's computer to a target service. These relay nodes, which act as packet forwarders are called Onion Routers (ORs), since Onion Routing[13] is used during the routing process. A user, who wants to utilize TOR, runs a TOR client on their computer, called TOR Proxy (TP). The TP contacts the TOR Directory Servers, which list all the available ORs, and then builds TOR circuits. Typically, a TOR circuit comprises of three ORs, the entry, the middle and the exit OR, but the system does not impose any constraints in the length of a TOR circuit. As long as the user transmits data, the TOR circuit remains functional. The information transmitted by the TP is encapsulated in TOR cells. A TOR cell is considered as the base information unit of transmission via TOR and it is 512 bytes long in size.

To provide stronger anonymity guarantees, the TOR system is designed so that any OR except the last one, is unable to identify its position in the TOR circuit[2]. On the other hand, to avoid eavesdropping, each TOR cell is routed

[1] At the time of writing this paper the number had reached about 2500 nodes (`http://torstatus.kgprog.com/`).

[2] Although the first one is also able to know its position by checking whether the IP address of the node before it is in the set of the Tor nodes. This set is available through the Directory servers [4].

using Onion Routing. That is, each TOR cell is multiply encrypted using symmetric cryptography. Each OR can decrypt only a single layer of the cell using its shared session key. Thus, the TP must encrypt each cell with all the shared session keys of the ORs that compose the TOR circuit.

The Packet Spinning Attack relies in two fundamental principles:

- Since the complete circuit is not known by every OR, circular circuits are not detectable.
- A legitimate OR will always spend some time in cryptographic operations.

The attack consists of two two phases, (*i*) the LOOP phase and (*ii*) the COMPROMISE phase. We continue by describing each phase in detail.

LOOP Phase. During the LOOP phase, an adversary attempts to keep a significant amount of legitimate ORs busy in spinning fraudulent packets. For an adversary to launch the attack in its simplest form, they need a malicious TP and a malicious OR. The malicious TP colludes with the malicious OR. The TP creates a TOR circuit which starts and ends at the malicious OR. The TP creates a packet, which it encrypts layer by layer using the shared symmetric session keys of the legitimate ORs composing the initiated TOR circuit, and then forwards it to the malicious OR. The malicious OR does not decrypt the packet but instead it immediately forwards it to the next legitimate OR of the circuit. The OR decrypts a layer of the packet and forwards to the next one, and so on. Finally, the packet completes a cycle and reaches the malicious OR completely decrypted. Upon receipt, the malicious OR drops the packet and re-injects the initial, fully encrypted packet, back into the circuit. This marks an artificial spin of a packet inside a TOR circuit. The same operation can be repeated indefinitely.

A schematic representation of the LOOP phase can be seen in Figure 1. To further amplify the attack, the malicious TP can build a series of loops, in various combinations, always using a single malicious OR, who is responsible for maintaining the loop.

It is vital to observe, that one malicious OR can keep multiple legitimate ORs busy. This is achievable for two reasons. First, the malicious OR spends much less time in packet routing, since it is not involved in any cryptographic operation. We evaluate this further in Section 4. Second, the malicious OR can be part of a circular circuit of arbitrary length. The default length of a TOR circuit is three ORs, but the protocol specification does not impose any constraints in building larger circuits. The ability of building TOR circuits of arbitrary length is also further explored in Section 4.

In addition, as we experimentally observe in Section 4 the difference in routing effort between the malicious OR and the legitimate ones increases as the packet size (the number of cells composing the initial packet) grows.

COMPROMISE Phase. When the attacker completes the LOOP phase, they are able to launch the COMPROMISE phase, in order to reveal anonymous communications. The adversary injects malicious ORs, which are not selected as

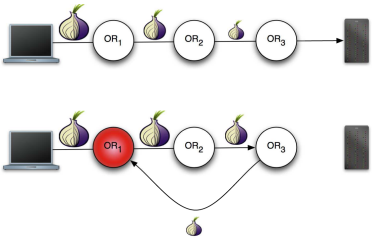


Fig. 1. Schematic representation of the packet spinning attack (LOOP phase). In the top part of the figure we depict the normal operation of a TOR circuit. Each OR decrypts a layer of the incoming packet and forwards it to the next one, until the packet reaches the final destination. In the lower part of the figure we depict the spinning packet attack. A malicious OR (solid cell), part of the circuit, injects again the initial packet in the circuit. The legitimate ORs continue to decrypt the packet in each spin, but the malicious one is not involved in any cryptographic operation.

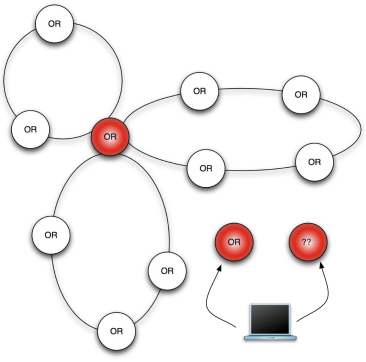


Fig. 2. Schematic representation of how an adversary can compromise the anonymity of a user by employing the packet spinning attack (COMPROMISE phase). In this figure, the adversary has injected three malicious ORs (solid cells) in the anonymizing system. One of them has built circular circuits with legitimate ORs in order to keep them busy, and two of them are free of resources in order to serve as routers for new TOR connections.

part of any *"spinning circuit."* That is, the injected ORs are idle and therefore they can be selected for legitimate TOR circuits with greater probability. As we investigate further in Section 4, even if legitimate ORs, which are part of a spinning circuit, are selected in the creation of a new TOR circuit, most likely they will not be able to join it. That is, if a legitimate TP selects ORs, which are occupied in spinning circuits, there is a great probability for the TP to receive a *timeout* from the selected ORs and continue building the circuits with new ORs. Sooner, or later the legitimate TP will select idle ORs like the ones injected in the system by the adversary. We schematically present the COMPROMISE phase in Figure 2.

4 Attack Evaluation

In this section we estimate the firepower of the packet spinning attack. We experimentally evaluate the attack magnitude by placing a malicious OR in a TOR overlay on two fronts:

– The overhead for a malicious OR to forward spinning packets versus the overhead of a legitimate OR to perform the same operation.
– The time required for a malicious TOR client to build arbitrary length TOR circuits.

Routing Overhead. The spinning packet attack is based primarily on the fact that a malicious OR may route packets faster than a legitimate one, because it is not involved in any cryptographic operation. The malicious OR is positioned in a circular TOR circuit and it is continuously injecting TOR cells inside the circuit, without decrypting the cells it receives (see Figure 1).

To measure the effort spent by a malicious OR to conduct the attack, in contrast to a legitimate OR, we conducted the following experiment. We placed three ORs on three hosts. Each host was running a single OR and all the three hosts were isolated from any external network traffic. One of the ORs was modified to be malicious. The same host that runs the malicious OR is also running a modified TP to create the circular circuit, as well as another 20 TOR processes and the directory servers needed for each OR to resolve the other available ORs. That is, the host running the malicious OR was significantly more loaded than the ones running the legitimate ORs. We used the default configurations of all ORs and we used the latest version of TOR (0.1.2.18) at the time we conducted the experiments.

We conducted several experiments for various numbers of spinning cells. In each run we measured the time needed for an OR - legitimate or malicious - to route the incoming cell. The time measured was from the point that the OR received a packet, up to the point the OR had sent the packet to the next hop of the circuit. For each experiment, the TP sends to the artificially made circuit a packet of specific size in terms of number of cells. Recall from Section 3 that the base information unit in TOR is the cell, which is equal to 512 bytes. That is, the TP sends packets that are multiple of one cell in length. From the moment the TP sends spinning packet to the circuit, the packet starts spinning with the assistance of the malicious OR and the TP is never again involved in the process. In Figure 3 we present the cost of routing the spinning cells for the legitimate and the malicious ORs. Observe, that for each OR, malicious or legitimate, the effort grows linearly to the packet size. In addition, the effort of the malicious OR is significantly lower to the legitimate ORs, and the difference increases as the cell number increases. More precisely, the difference in the effort can grow from approximately 60% to nearly 85%. In Table 1 the relative percentage of the difference during the routing effort is listed for each experiment.

As far as the spins achieved by each spinning packet are concerned, we show the time required for a series of cell relays in Figure 4. Each OR can spin a cell at a maximum rate of 25 relays per second.

Circuit Building. One fundamental property of the packet spinning attack is that a malicious OR can occupy several legitimate ORs, by making circular circuits of arbitrary length. Recall that the default length of a TOR circuit length is three, but the protocol does not impose any constraints for larger circuits.

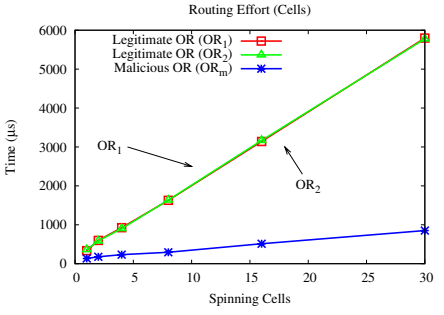


Fig. 3. The cost in terms of time for legitimate ORs to forward spinning packets in contrast with the effort of a malicious OR.

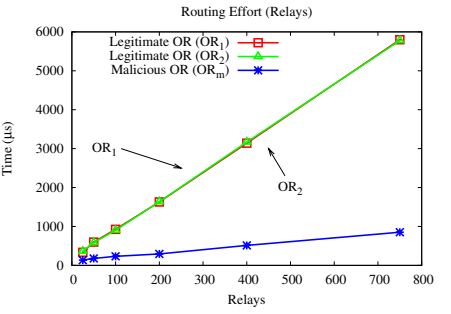


Fig. 4. The cost in terms of time for legitimate ORs to forward spinning packets in contrast with the effort of a malicious OR.

Table 1. The percentage difference of the routing effort spend by the malicious OR (OR_m) relatively compared with the routing effort spend by the legitimate ORs, (OR_1, OR_2). Routing effort is recorded in μs, which is the period of time required for an OR to send the packet to the next hop in the circuit after it has successfully received it.

OR_1	OR_2	OR_m	$\frac{OR_1 - OR_m}{OR_1}$	$\frac{OR_2 - OR_m}{OR_2}$
336.199	361.403	133.136	60%	63%
598.276	584.898	176.997	70%	70%
926.444	901.521	232.435	75%	74%
1629.16	1635.22	294.156	82%	82%
3140.73	3167.97	513.536	84%	84%
5798.09	5777.1	850.997	85%	85%

In Figure 5 we present real world experiments for the creation of long TOR circuits over time. Notice that, even though the time needed for the circuit creation increases exponentially in terms of the circuit length, we were able to create TOR circuits of more than 30 hops. This means that using one malicious OR we would be able to keep busy more than 30 legitimate ORs.[3]

5 Compromising Anonymity

Based on the results we highlighted in Section 4, we proceed to explore how an actual packet spinning attack can compromise the anonymity of users that utilize a real anonymizing system, namely TOR.

Experimental Setup. To conduct the experiments we used the `TorFlow` [2] package which is a complementary package of the TOR project [1]. `TorFlow` is written in `Python`, and it is composed of a series of scripts, which utilize the TOR control channel to communicate with active ORs. The TOR software supports a

[3] In reality, we can do even better. We can use one malicious OR to issue circular circuits with several other ORs, as it is depicted in Figure 2.

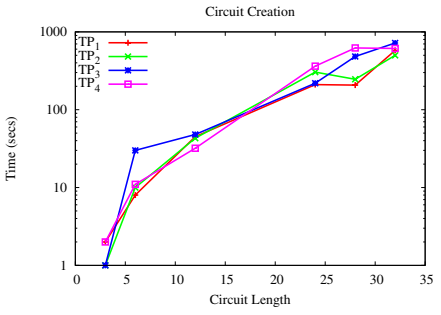


Fig. 5. The time needed for a malicious OR to issue TOR circuits of arbitrary length

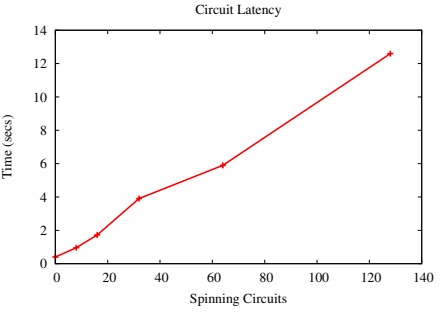


Fig. 6. Time latency for building additional circuits over an existing set of Tor nodes

protocol for OR instrumenting. A client can connect to a specific port, bound to the control channel of an OR and give commands in a request-response fashion. Some operations that are allowed are: building circuits, attaching streams to circuits and querying an OR for various statistics.

All experiments were held in a private and isolated from the rest of the Internet TOR overlay. Using `TorFlow` we developed scripts that were creating a malicious OR which was part of the overlay. The rest of the ORs were kept intact in terms of software modifications. The only modification we did was the bandwidth constraints of the legitimate ORs. We set all legitimate ORs to be bounded to 1 Mbit per second. We did this to shorten the times it took to conduct our experiments. As we will discuss later, an adversary could launch a similar attack in a TOR overlay that is composed of ORs that experience greater bandwidth, by injecting more spinning cells in the artificial made loops.

Our strategy was the following. We were starting a TOR overlay of a variable number of ORs and we had a `TorFlow`-based Python script that was acting as a malicious TP, by instrumenting a malicious OR, and a legitimate TP that was trying to access the Internet using our isolated TOR. When our scripts weren't running, the legitimate TP could access the Internet using our TOR in a normal fashion. On the other hand, as we explore in detail in the rest of this section, when our scripts were running, the legitimate TP was experiencing side effects that could potentially lead to anonymity compromising.

Packet Spinning Effects. The first evident behavior experienced by the legitimate TP was the inability of circuit building in time. When a TP tries to access the Internet using TOR, it first selects three ORs and then tries to build the circuit telescopically. That is, it first contacts and establishes connection with the entry router, it then expands the circuit by tunneling the requests through the entry router until the circuit is created. However, there is a timeout for the creation of circuits, which is set by default at 60 secs. In addition, ORs that were part of spinning circuits were unable to process the circuit creation requests fast enough and thus most of the circuit creation operations issued by the TP were failing.

In order to demonstrate this side effect more clearly, we contacted an experiment with a TOR overlay that was running on a single host. We eliminated out all network latencies since all communications between the TOR processes were local. Using our scripts we created an artificial loop of length five (the malicious OR was the first and last router and the legitimate ones were the three in the middle) and we started spinning cells inside. We then forced another script to create the same circuit. We proceeded in adding more spinning circuits and trying to build a new circuit over them. In Figure 6 we present our results.

Notice, that when there are no spinning circuits the circuit creation is almost spontaneous. However, as the spinning circuits increase, the circuit creation becomes a long process (recall that we have eliminated all network latencies, since the overlay runs in a single host) exceeding a period of 10 seconds.

6 Countermeasures

An anonymizing system that is based on a series of in-between relay nodes, and on TOR like circuits, is vulnerable to a packet spinning attack since it permits the creation of circular circuits. An adversary could utilize these circuits, having a malicious relay node that is not taking part in cryptographic operations and thus it has time to flood the circular circuit with fraudulent packets. We carried out all this analysis, using a real anonymizing system, TOR.

An obvious countermeasure would be to embed information of a circuit in all participating relayers. This would prevent the creation of circular circuits, but subsequently would decrease the anonymity level provided by the system. We are against any countermeasure that degrades the anonymity level of existing systems.

Instead of preventing circular circuits, our solution aims at reducing the effects of artificial loops in an anonymizing overlay. We propose existing anonymizing systems to employ *Tree Based Circuits (TBCs)*. More precisely, instead of having serial circuits, like the ones used in TOR, we propose that circuits will expand from the entry node in a tree fashion targeting the final destination. In Figure 7 we depict a TBC. The entry node issues two connections with two middle nodes and each of the middle nodes issues two connections with two exit nodes. In this example, one exit node is shared between the two middle nodes, but this is not obligatory. The dashed lines present OR connections that are ready to be utilized and the solid lines present an active TOR stream.

Introducing TBCs in an anonymizing system like TOR raises some important questions. *How fast a TBC can be constructed?* Recall, that TOR is used for low-latency communications. *Does it degrades the level of anonymity?* Notice, that by employing TBCs will be impossible to make circuits with many levels in terms of hops, since trees grow exponentially. *Is a TBC vulnerable to packet spinning?*. In the rest of this section, we address each of this question in detail.

How fast a TBC can be constructed? Currently, TOR builds four circuits on startup. This is done for redundancy. Each OR maintains these four circuits and it is able to use any of them upon a circuit failure. Our proposal, follows the same

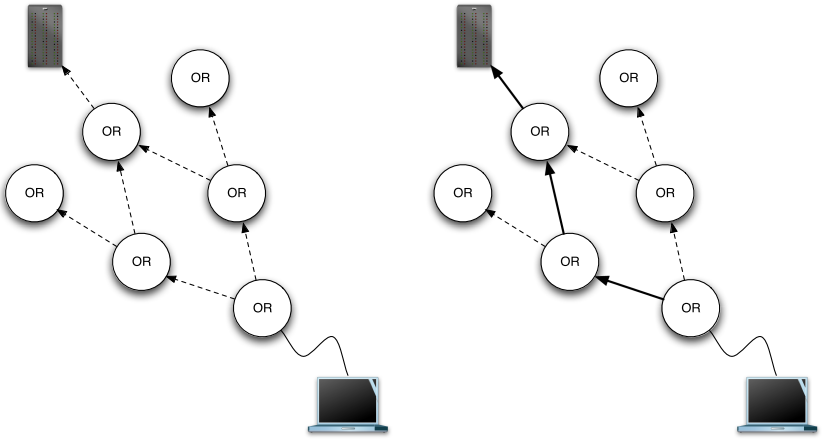


Fig. 7. Proposed solution. Tree based TOR circuits that are not vulnerable to circular circuits.

logic, but instead of creating four distinct circuits, we create a TBC. A TBC has many circuits that can be used as alternatives upon a circuit failure. Moreover, the TBC can be created asynchronously; the client does not need to wait for the whole TBC creation in order to start transmitting information. The TP remains responsible for the circuit creation, but for each expand operation, the circuit is expanded towards multiple directions giving, schematically, the impression of a tree structure.

Does it degrades the level of anonymity? The ability of building large circuits in terms of hops gives the impression of higher anonymity, since the packet is relayed more times, although the latency, for the same reason, is increased. By employing TBCs the relay node number increases, but the latency does not, since the hops from the entry node to the exit one are kept at a low level (again there is no constraint for the depth of the TBC, but it is evident that large TBCs are an expensive operation, since trees grow exponentially). Essentially, the relay nodes involved in a transaction are of the same magnitude as of the relay nodes involved in current TOR circuit, but the relay nodes involved in the whole protocol negotiation are many more. The additive cost of a TBC in contrast with a plain TOR circuit is that each OR has to maintain some state in order to correctly route requests back to the TP.

Is a TBC vulnerable to packet spinning? A TBC can not contain loops, but again an adversary can place some malicious nodes in order to create TBCs that send requests back to the entry node and in this way create artificial loops. However, the adversary has to own more than one node in order to compromise a circuit, but, more of importantly, the users can escape more easily from loops by routing their requests using TBCs instead of plain circuits. A TBC gives the user more alternative circuits to the final destination, which in turn decrease the probability of encountering nodes that are overwhelmed by spinning packets in a circular circuit.

7 Conclusion and Future Work

In this paper we presented a novel attack against modern anonymizing systems, in which a series of relay nodes route cryptographically wrapped packets. The attack is based on inserting a malicious node in an anonymizing overlay, that is able to construct circular circuits and forces packets to spin indefinitely inside those loops. In this fashion an adversary can keep legitimate routers busy while at the same time they can inject their own, unloaded, malicious nodes. Since these nodes are not kept busy, they have a higher probability of being selected by users wanting to utilize the anonymizing system. This way, new circuits are more likely to contain malicious nodes, and the anonymity of the user can be compromised.

We evaluated our attack using a real-world TOR system and our evaluation showed that such an attack is feasible. Finally, we came up with a method to counter packet spinning attacks and proposed Tree-Based Circuits. We showed that TBCs can be constructed relatively fast, they do not degrade the anonymity properties of the system and they are not vulnerable to packet spinning. Part of our future work is to implement TBCs in the TOR system and evaluate their performance, as well as further examining if a TBC is vulnerable to similar attacks like the one presented in this paper.

Acknowledgments

We thank the anonymous reviewers for their valuable comments. Vasilis Pappas, Elias Athanasopoulos, Sotiris Ioannidis, and Evangelos P. Markatos are also with the University of Crete. This work was supported by the Marie Curie Actions - Reintegration Grants project PASS. Elias Athanasopoulos is also funded from the PhD Scholarship Program of Microsoft Research Cambridge.

References

1. The TOR Project, http://www.torproject.org/
2. TorFlow, https://www.torproject.org/svn/torflow/README
3. Back, A., Möller, U., Stiglic, A.: Traffic analysis attacks and trade-offs in anonymity providing systems. In: Moskowitz, I.S. (ed.) IH 2001. LNCS, vol. 2137, pp. 245–257. Springer, Heidelberg (2001)
4. Bauer, K., McCoy, D., Grunwald, D., Kohno, T., Sicker, D.: Low-resource routing attacks against tor. In: Proceedings of the Workshop on Privacy in the Electronic Society (WPES 2007), Washington, DC, USA (October 2007)
5. Berthold, O., Pfitzmann, A., Standtke, R.: The disadvantages of free MIX routes and how to overcome them. In: Federrath, H. (ed.) Designing Privacy Enhancing Technologies. LNCS, vol. 2009, pp. 30–45. Springer, Heidelberg (2001)
6. Borisov, N., Danezis, G., Mittal, P., Tabriz, P.: Denial of service or denial of security? How attacks on reliability can compromise anonymity. In: Proceedings of CCS 2007 (October 2007)

7. Chaum, D.: Untraceable electronic mail, return addresses, and digital pseudonyms. Communications of the ACM 4(2) (Febuary 1981)
8. Clarke, I., Sandberg, O., Wiley, B., Hong, T.W.: Freenet: A Distributed Anonymous Information Storage and Retrieval System. In: Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability, July 2000, pp. 46–66 (2000)
9. Danezis, G.: The traffic analysis of continuous-time mixes. In: Martin, D., Serjantov, A. (eds.) PET 2004. LNCS, vol. 3424, pp. 35–50. Springer, Heidelberg (2005)
10. Danezis, G., Dingledine, R., Mathewson, N.: Mixminion: Design of a Type III Anonymous Remailer Protocol. In: Proceedings of the 2003 IEEE Symposium on Security and Privacy, May 2003, pp. 2–15 (2003)
11. Dingledine, R., Mathewson, N., Syverson, P.: Tor: The second-generation onion router. In: Proceedings of the 13th USENIX Security Symposium (August 2004)
12. Freedman, M.J., Morris, R.: Tarzan: A Peer-to-Peer Anonymizing Network Layer. In: Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS 2002), Washington, DC (November 2002)
13. Goldschlag, D.M., Reed, M.G., Syverson, P.F.: Hiding routing information. In: Information Hiding, pp. 137–150 (1996)
14. Mathewson, N., Dingledine, R.: Practical traffic analysis: Extending and resisting statistical disclosure. In: Martin, D., Serjantov, A. (eds.) PET 2004. LNCS, vol. 3424, pp. 17–34. Springer, Heidelberg (2005)
15. Murdoch, S.J., Danezis, G.: Low-cost traffic analysis of Tor. In: Proceedings of the 2005 IEEE Symposium on Security and Privacy, May 2005. IEEE Computer Society Press, Los Alamitos (2005)
16. Nambiar, A., Wright, M.: Salsa: A Structured Approach to Large-Scale Anonymity. In: Proceedings of CCS 2006 (October 2006)
17. Raymond, J.-F.: Traffic Analysis: Protocols, Attacks, Design Issues, and Open Problems. In: Federrath, H. (ed.) Designing Privacy Enhancing Technologies. LNCS, vol. 2009, pp. 10–29. Springer, Heidelberg (2001)
18. Reiter, M., Rubin, A.: Crowds: Anonymity for web transactions. ACM Transactions on Information and System Security 1(1) (June 1998)
19. Rennhard, M., Plattner, B.: Introducing MorphMix: Peer-to-Peer based Anonymous Internet Usage with Collusion Detection. In: Proceedings of the Workshop on Privacy in the Electronic Society (WPES 2002), Washington, DC, USA (November 2002)
20. Sherwood, R., Bhattacharjee, B., Srinivasan, A.: P5: A protocol for scalable anonymous communication. In: Proceedings of the 2002 IEEE Symposium on Security and Privacy (May 2002)
21. Snader, R., Borisov, N.: A tune-up for Tor: Improving security and performance in the Tor network. In: Proceedings of the Network and Distributed Security Symposium - NDSS 2008, February 2008, Internet Society (2008)
22. Zhuang, L., Zhou, F., Zhao, B.Y., Rowstron, A.: Cashmere: Resilient Anonymous Routing. In: Proc. of NSDI, Boston, MA, May 2005, ACM/USENIX (2005)

Appendix: Technical Discussion

Routing in Tor

Each Tor node has a routing table that contains entries in the form: (source connection, source circuit id) - (destination connection, destination circuit id)[4]. Upon a circuit establishment, each node adds an entry in its routing table to forward cells for that circuit. In order for the last node (exit node) to know its position, the routing entry for the specified circuit does not have a destination connection (is has a NULL value of destination connection). After the circuit establishment, Tor nodes are able to route cells in a circuit. The pseudo-code of the function executed upon receiving a new cell is shown in Figure 8.

A simple way to make a cell spin in a circular circuit is to change the last node's routing entry's destination connection from NULL to the connection with the second node in the circuit. That way whenever the last node receives a packet it will forward it again to the second node in the circuit.

```
function receive_cell(cell c)

  decrypt_one_layer(c)

  if (is_recognized(c)
    //do exit node stuff
    //...
  else

    next_conn, next_circ_id = get_route_info(c)

    if (next_conn)
      c.circ_id = next_circ_id
         send_cell(c, next_conn)
    else
      //circuit stops here bu the cell wasn't recognized
      drop_cell(c)
```

Fig. 8. The receive cell Tor's function pseudocode

Spin in Tor Implementation

Keeping the previous in mind, we altered the source code of Tor in order to implement the cell spin. The procedure to make a cell spin in in a Tor circuit using a colluding TP and an colluding OR is comprised by the following steps:

1. `Establish a circular circuit.` Although Tor's node selection algorithm never selects a node twice (as an entry and as en exit) we used the control component of Tor (through `TorFlow` [2]) to explicitly select the nodes for a circuit. That way, we create a circular circuit with out colluding OR placed at entry and exit positions.

[4] Connection denotes a TLS connections with other Tor nodes. Many Tor circuits can be multiplexed in a single connection.

2. `Inform the colluding OR.` In order to get the cell spin we have to inform the colluding OR to change its routing table. So, the first thing to do is to get the destination connection and destination circuit id pair that forwards cells to the second OR. This is done by sending a cell (containing a special message) encrypted only with the entry node's key to the circuit. That way our colluding OR (as an entry node) recognizes it, keeps the destination connection and circuit id and forwards it down the circuit.
3. `Change OR's routing table.` The previous cell that our colluding OR forwarded will end up again to it but this time it wont be recognizable. So, our colluding node will get an unrecognizable cell that stops there. That time it will suppose that this cell is the one it forwarded before and will set the destination connection and circuit id of the current routing entry to the values kept at step 2.

Behavior-Based Network Access Control: A Proof-of-Concept

Vanessa Frias-Martinez, Salvatore J. Stolfo, and Angelos D. Keromytis

Computer Science Department, Columbia University
{vf2001,sal,angelos}@cs.columbia.edu

Abstract. Current NAC technologies implement a pre-connect phase where the status of a device is checked against a set of policies before being granted access to a network, and a post-connect phase that examines whether the device complies with the policies that correspond to its role in the network. In order to enhance current NAC technologies, we propose a new architecture based on *behaviors* rather than *roles* or *identity*, where the policies are automatically learned and updated over time by the members of the network in order to adapt to behavioral changes of the devices. Behavior profiles may be presented as identity cards that can change over time. By incorporating an Anomaly Detector (AD) to the NAC server or to each of the hosts, their behavior profile is modeled and used to determine the type of behaviors that should be accepted within the network. These models constitute behavior-based policies. In our enhanced NAC architecture, global decisions are made using a group voting process. Each host's behavior profile is used to compute a partial decision for or against the acceptance of a new profile or traffic. The aggregation of these partial votes amounts to the model-group decision. This voting process makes the architecture more resilient to attacks. Even after accepting a certain percentage of malicious devices, the enhanced NAC is able to compute an adequate decision. We provide proof-of-concept experiments of our architecture using web traffic from our department network. Our results show that the model-group decision approach based on behavior profiles has a 99% detection rate of anomalous traffic with a false positive rate of only 0.005%. Furthermore, the architecture achieves short latencies for both the pre- and post-connect phases.

Keywords: Network Access Control Technologies, Intrusion Detection Systems.

1 Introduction

Network Access Control (NAC) technologies manage the access of devices to a network and mitigate against inside threats within a network. This is accomplished by implementing a two-tier strategy: the pre-connection and the post-connection phases. The pre-connection phase checks whether a device attempting to connect to a network complies with a set of policies. These policies typically include checking the status of the antivirus (AV) software in the device and whether or not the required patches for the OS are installed. If the device is not up-to-date, it is either quarantined or rejected from connecting to the network. The post-connection phase controls whether the policies (AV software, patches) are still being complied with by the network hosts. It may also

T.-C. Wu et al. (Eds.): ISC 2008, LNCS 5222, pp. 175–190, 2008.

include traffic monitoring meant to detect any anomalous traffic using Signature-based or Anomaly-based Detection Systems (AD).

The current generation of NAC technologies rely on the use of fixed roles in the network. A list of roles is initially declared manually using a pre-determined set of characteristics. Devices are then provided with roles in the network that can only be changed manually. These roles are not only used to decide what devices are granted access to the network, but also to monitor what type of actions are allowed for each device. In reality, networks are very dynamic environments where devices may change roles or new roles may have to be created. Unfortunately, updating and defining new roles manually becomes very demanding and highly inefficient as time elapses. Ideally, we seek a solution that can define and update roles automatically without the inception of a human in the loop.

In this paper we introduce a new Behavior-Based Network Access Control architecture, *BB-NAC*, in which the behavior profiles of network hosts modeled by an AD are used to automatically compute and update behavior-based policies to enhance security. This new strategy enhances current NAC technologies by accounting for host behavior and its changes. The use of behavior profiles allows us to automatically conform to changes in behavior and update security policies without human intervention. In our proposed architecture, AD sensors are used to model the profile of the hosts in the network. Profiles are communicated by devices as a representation of their typical behavior. As behaviors change, updated models computed by the AD are captured as new behavior profiles. These behavior profiles can be used as a declaration of intent of behavior. In this manner, devices that drift from their profile are either under attack or have lied about their typical behavior.

In terms of deployment, BB-NAC can be implemented either as an agent NAC architecture where the AD is installed directly on each of the hosts in the network, or alternatively as an agentless NAC architecture using a unique AD installed on the NAC server. Here a NAC server denotes a server that sits on the edge of the network and listens to incoming and outgoing traffic. In an agent NAC architecture, each host computes its behavior profile and communicates it to the NAC server. In an agentless NAC architecture, the NAC server itself models the individual behavior of each host in the network and stores its profile locally. By modeling each profile individually, rather than as a group, profiles of similar behavior can be clustered together and differentiated from other types of behavior. As an aside, we note that our architecture can be applied to networks without a central control like Mobile Ad-hoc Networks (MANETs) by eliminating the NAC server from the architecture. The latter is beyond the scope of this paper. In the following sections, we present a generalized description of the architecture that can be implemented either as an agent or as an agentless version with minor modifications.

In terms of execution, the BB-NAC architecture performs pre-connection and post-connection checks based upon a group decision made by the NAC server using the profiles of the devices already in the network. During pre-connection, a device attempting to access the network presents its profile to the NAC server that conducts a voting process among the stored profiles of the hosts already in the network to reach an access control decision. During post-connection, the validity of the traffic exchanged is similarly voted by the profiles in the network. If the group decision is positive, the

device is granted access to services. Otherwise, the device is either quarantined or rejected from accessing a service. Individual hosts do not participate actively in the voting process, but rather it is the NAC server that conducts the voting among the group of individual models (profiles) stored on the NAC server. Throughout the paper, we refer to decisions made in this manner as *model-group decision*. Such model-group decision process increases the survivability of the network by minimizing the influence of malicious profiles. As mentioned previously, profile clustering is introduced to attain a more fine-grained definition of network behavior. In this manner, only hosts in clusters with sufficient knowledge participate in the voting process. Below we summarize the main contributions of this paper:

- A new technique to automatically learn and update access control policies using behavior. This approach enhances existing NAC technologies by providing, to the best of our knowledge, the first *behavior-based* network access control.
- A novel access control model based on a model-group decision process. Individual host's behavior profiles stored on the NAC server are used to compute partial decisions. The aggregation of these partial votes amounts to the model-group decision.
- An architecture resilient to attacks. The access control model continues to work even after allowing a certain number of malicious devices into the network.
- An implementation for agent or agentless NAC technologies. By installing an AD on the NAC server or on the hosts, the model-group decision process is conducted by the NAC server in similar fashion.
- An architecture that is independent of the type of AD sensor used: content ADs, volumetric ADs, or others.

In Section 2 we describe related work. Section 3 introduces the BB-NAC architecture. Section 4 shows experimental results and latency analysis of our architecture. Finally, Section 5 covers conclusions and future work.

2 Related Work

To the best of our knowledge, we are the first to introduce *behavior model exchange* as a security feature. Possibly the closest concept to our approach was developed by Necula and Lee [5], [4] and [6] in their Proof-Carrying Code (PCC). However, our approach differs in the fact that behavior can be automatically learned from observation, whereas proofs are specified by hand. Furthermore, our architecture proposes the exchange of behavior models instead of safety proofs. *Cooperative Anomaly Detection Sensors* have been explored in WORMINATOR [8], COSSACK [7] and CATS [3] where a distributed environment shares alerts to strengthen each individual local security capabilities. We implement the concept of cooperation in the model-group decision process by allowing each host to participate in the access control decision rather than just sharing alerts.

A number of NAC technologies are currently available in the market. The *Trusted Network Connect* (TNC) is an initiative of the *Trusted Computing Group* that proposes a non-proprietary standard to enable the enforcement of security policies on endpoints. *Cisco Network Module for Integrated Services Routers* offers an agentless solution authenticating, authorizing and remediating devices connected wired or wirelessly to the

network. The *Cisco Profiler* executes an in-depth control of the endpoint devices of the network by passively monitoring their traffic. The *Network Access Protection* (NAP) platform from *Windows*, provides a client and server-side platform (Longhorn) to implement policy validation, network access limitation, automatic remediation and ongoing compliance. Compared to all other previous NAC technologies, our architecture uses behavior computed by an AD instead of roles (host posture) as a security feature.

3 The BB-NAC Architecture

We start with the conjecture that behavior modeled by an AD can be used as a means to enhance and automate security enforcement in a NAC architecture. We assume that profiles or behavior models represent the typical behavior of a device. As opposed to roles, profiles can be automatically computed and updated by an AD as a device changes behavior over time. In our architecture, devices initially present their profiles to the NAC server prior to entering a network. Devices are also required to present a *bad model* that represents a collection of all previously seen bad attacks modeled using the same AD. The bad model measures the amount of knowledge the device has about known bad behaviors and might be considered a generalization of the set of rules or signatures used in a standard AV.

BB-NAC then follows the two-tier strategy commonly used in NAC architectures except that the *pre-connect* and *post-connect* phases are both based on a model-group decision process conducted on the NAC server i.e., an alert is raised whenever a set of profiles agree on the access control decision being made. Furthermore, the access control policies in the NAC are computed and updated automatically as the ADs compute new models. The *pre-connect phase* checks whether a device entering the network has up-to-date malware knowledge. Devices that do not have sufficient malware knowledge are quarantined or rejected from entering the network altogether. On the other hand, the *post-connect phase* is responsible for a continuous check of the traffic exchanged by the hosts in the network. The two-tier strategy is applied on a per-port basis. In the case of multiple ports, the two-tier strategy is executed separately for each individual port. The device is accepted only when it is deemed normal for all ports. Next, we describe the pre- and post-connect phases in more detail.

3.1 Pre-connect Phase

The pre-connect phase is responsible for checking whether a device attempting to enter the network has sufficient malware knowledge. During this phase, a device presents its behavior profile as well as its bad model to the NAC server. If the device is coming from a different network, the profiles presented are the ones modeled by the AD during its previous interactions. Otherwise, if the device is brand new, we assume that it is equipped with a *vanilla profile* or that it is given one by the network administrator prior to starting its interaction. Next, the NAC checks whether the device's bad profile contains sufficient malware information to be accepted to the network. This step is similar to conventional NAC approaches where the status of the AV is checked to determine whether it is up-to-date. However, in our solution the access control decision is based

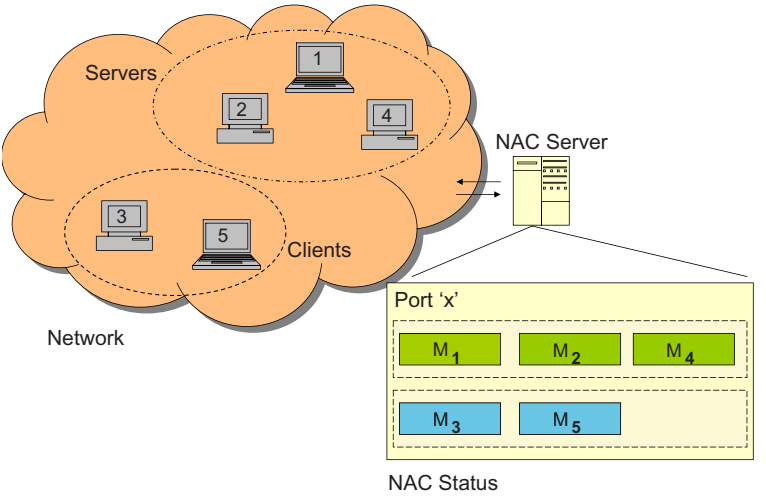


Fig. 1. Basic setup of the BB-NAC architecture. The NAC server stores the behavior profiles and bad models of each host in the network. It also stores the cluster information. Here, $M_i = \{P_i, B_i\}$.

on the group knowledge of malware among the hosts already in the network. This is unlike current NAC architectures where the amount of malware knowledge required is manually set up as a policy.

In order to attain a more accurate access control, only hosts with similar profiles to the one attempting to enter the network will be involved in the access control decision. The NAC server divides the devices into clusters representing different behaviors. These individual clusters are then responsible for the access control decision. If the device is accepted, the NAC server will add its profile to the corresponding cluster. For simplicity, in this paper we assume that the clusters of behavior are formed based on the *declaration of nature* provided by the device itself i.e., a device declares itself to be a *client* or a *server*. As a rule, it is required that a device of the same type already exists in the network. Obviously, it may be the case that a device lies about its true nature. However, if a device starts behaving anomalously, it will be detected in the post-connect phase.

Figure 1 shows the basic setup of our architecture. As can be seen, for any given *Port 'x'* there are two differentiated clusters: one for clients and one for servers. For each cluster, the NAC server stores both the profile and the bad model of its host members. We use M_i to denote the set of behavior profile (P_i) and bad model (B_i) for each host i i.e., $M_i = \{P_i, B_i\}$. The behavior-based access control policy is determined by whether or not the device's knowledge of malware is considered sufficient by the members of the cluster of identical nature. In short, each of the bad profiles in the cluster participates in a voting process to make an access control decision defined as:

$$V = (\sum_{k=1}^{n} v_i)/n \tag{1}$$

$$v_i = 1,\ if\ \ B_i \subset B_{device}$$
$$v_i = 0,\ if\ \ B_i \supset B_{device} \tag{2}$$

where n denotes the number of hosts that vote, where each vote v_i equals 1 when the new device knows at least as many bad attacks as host i, and $v_i = 0$ when the new device knows fewer attacks that host i. V represents the fraction of hosts in the cluster that consider the device's bad model has sufficient malware knowledge. The driving principle behind this calculation is a quantitative measurement that can grant or deny entrance to the network based on the agreement of a certain percentage of network host profiles. It may be the case that a group of malicious profiles collude to manipulate the vote. However, our architecture can withstand such attacks as long as the number of malicious profiles in the network does not dilute the percentage of agreement required among host profiles. In Section 4, we describe the impact of possible attacks on our architecture.

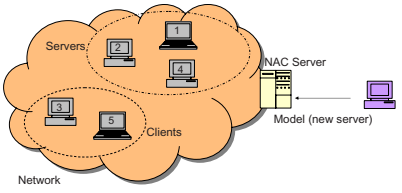
Figure 2 depicts the voting process. At *Step 1*, the NAC server listens to a new server attempting to connect to the network. This new server presents its profile and bad model to the NAC server. During *Step 2*, the NAC server conducts the voting process among bad models in the cluster to determine whether the malware knowledge is sufficient. Finally at *Step 3*, the accepted server is added to the cluster of servers and its profile and bad model are in the NAC server. In terms of deployment, the voting process is always conducted by the NAC server using the stored host profiles for both the agent and agentless versions of the architecture.

3.2 Post-connect Phase

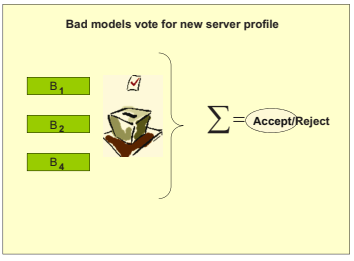
The post-connect phase performs a continuous check on the traffic being exchanged by the hosts in the network. The goal is to guarantee normalcy of behavior in the network. In our architecture, this is achieved by using the profiles of each individual host in the network that are stored in the NAC server. Armed with these profiles, BB-NAC determines whether or not the traffic is considered anomalous using a model-group decision process. The post-connect phase makes use of the clusters computed in the pre-connect phase. Profiles of similar behavior are clustered together so that only profiles akin to the source or destination of the traffic participate in the decision of traffic normalcy. Our architecture conducts a voting process where each profile votes for or against the normalcy of the observed traffic. The voting process is defined as:

$$V = (\sum_{k=1}^{n} P_{k,d}(t))/n \tag{3}$$

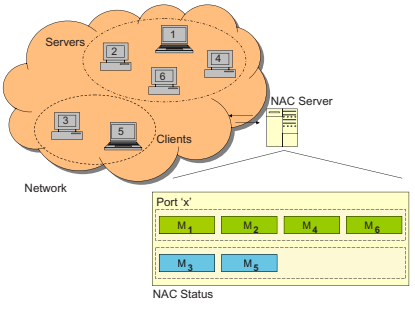
where $P_{k,d}$ represents the behavior profile of host k for direction d (ingress or egress) in a cluster with n hosts. Because the traffic can be analyzed either at a packet or flow level, t denotes the granularity (packet or flow) at which the traffic is tested against the AD profiles. The output of $P_{k,d}(t)$ equals 1 if the traffic unit is considered normal by $P_{k,d}$ and otherwise 0 if it is considered anomalous. V represents the fraction of hosts in the network that consider the traffic unit to be normal. In Section 4, we discuss the impact



(a) Step 1: New server presents its model to NAC server.



(b) Step 2: NAC server conducts the voting process.

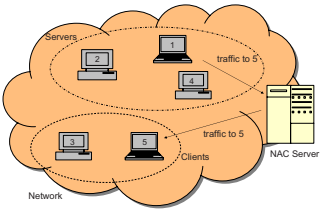


(c) Step 3: New server is accepted and NAC status is updated.

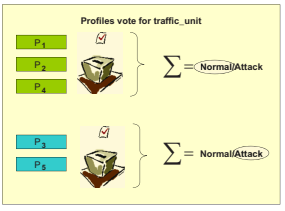
Fig. 2. Schematics of the Pre-connect Phase

of malicious devices trying to manipulate the voting process. Another key ingredient of the post-connect phase is that the observed traffic is used to compute new behavior profiles for each of the hosts as time elapses. If the observed traffic is considered normal by the cluster, it is used to compute a new profile for the members involved in the exchange. On the other hand, if the traffic is deemed anomalous, it is used to update the bad models of the hosts in the cluster. These new computations *automatically update the pre-connect and post-connect security policies*. Issues concerning concept drift are addressed in Section 4.3. In terms of deployment of the agent version of the architecture, new profiles are computed by the hosts and communicated to the NAC server which stores them locally. In an agentless version, on the other hand, the NAC server itself computes the new profiles and stores them locally. In both versions, the voting process is always conducted by the NAC server among the profiles stored locally.

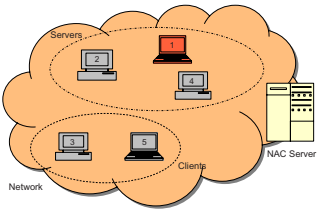
Figure 3 depicts a possible scenario during the post-connect phase. In this setting, traffic is exchanged from host 1 to host 5 (*Step 1*). During *Step 2*, the NAC server implements two checks. First, it checks whether the output traffic of host 1 is considered normal by host 1 and all the other profiles in its cluster. Second, it checks whether the input to host 5 is considered normal by host 5 and all the other profiles in its cluster. In this instance, the second check reveals an attack. As presented, this two-layer check makes the architecture more resilient to insider threats. Next, in *Step 3*, the source is



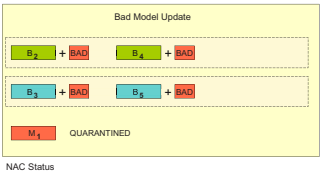
(a) Step 1: Traffic is exchanged from host 1 to host 5.



(b) Step 2: NAC server conducts two voting processes. The cluster of servers votes on the output traffic of host 1 and the cluster of clients votes on the input traffic to host 5.



(c) Step 3: Traffic is deemed anomalous and the sender is placed in quarantine.



(d) Step 4: The NAC status is updated with the detected attack. Sender is placed in the quarantine queue.

Fig. 3. Schematics of the Post-connect phase

placed into quarantine to determine whether or not it is infected. Lastly in *Step 4*, the NAC server updates the malware knowledge of the bad models and updates the queue with the hosts in quarantine.

4 Experiments and Evaluation of the Architecture

For initial evaluation of our architecture, we collected web traffic from the Computer Science department network at Columbia University for a period of three weeks. We only considered IPs within the local network and divided them into two clusters: *servers* and *clients*. The nature of the machines is known from the collection of local IPs kept by the department. Since the pre- and post-connect phases are executed separately for each individual port, we chose port 80 to validate our architecture. Experiments for other ports would be executed in a similar fashion.

For our proof-of-concept experiments, we modeled the profiles of all the webservers (a total of four) in the department using the anomaly detection sensor Anagram [11]. Anagram is a content anomaly sensor that models a mixture of n-grams to detect suspicious network packet payloads. The profile content models are saved as Bloom filters

[1] which are space and privacy preserving data structures consisting of a vector of 0s and 1s. In general, Bloom filters suffer from false positives but not from false negatives. Furthermore, Bloom filters can be exchanged among devices and NAC servers minimizing the risk of privacy violation. Although we only used Anagram, it is important to note that our architecture is flexible enough to allow any AD to be used.

In order to compute the profiles, we used between 370K and 700K clean training packets obtained from the first two weeks of the collected traffic. These profiles were trained until stability was reached i.e., the point in time when the ratio of observed n-grams divided by the total number of n-grams was below a threshold. In addition, the *bad models* for each webserver were computed following the technique described by Wang et al. [11]. Each profile was trained with signature content from Snort rules [9] and with 600 virus samples collected from *vxheavens* [10]. In terms of actual deployment in a network, the BB-NAC architecture would have to be installed in a NAC server at the edge of the network. In addition, the NAC server would have to store the profiles and bad models of each of the webservers in the *server cluster*.

4.1 Evaluation of the Pre-connect Phase

In order to add artificial diversity to our network and create a more meaningful proof-of-concept experiment, we compute a bad model for each of the four webservers in such a way that each bad model contains 10% less malware knowledge than its previous model. Such computation is meant to simulate the behavior of users that have forgotten to update their AVs one or multiple times. Therein, *server1*'s bad model contains all the collection of Snort rules and virus samples, *server2*'s bad model contains 10% less than *server1* randomly excluded from the collection, *server3*'s bad model contains 10% less than *server2* randomly excluded from the collection, and *server4*'s bad model contains 10% less than *server3* randomly excluded from the collection. While in a real network percentages may vary from server to server, a value of 10% was arbitrarily chosen for this proof-of-concept experiment to validate that the access control policy functions properly.

In our setup, we assume that three out of the four webservers are members of the network and that the fourth device (self declared as a server) attempts to enter the network. This configuration allows us to evaluate three different scenarios: (i) Only one server agrees on the acceptance of the new server (one out of three, 33%), (ii) Two servers (66%) agree on the acceptance of the new server, and (iii) All three (100%) servers agree to accept the new server to the network. Given the fact that the bad models are represented as Bloom filters, each vote in the voting process is calculated as follows:

$$\begin{aligned} v_i &= 1,\ if\ \ |B_i \wedge B_{device}| \ / \ |B_i| = 1 \\ v_i &= 0,\ if\ \ |B_i \wedge B_{device}| \ / \ |B_i| < 1 \end{aligned} \tag{4}$$

where v_i is the vote of server i, B_i is the bad model of server i, and B_{device} is the bad model of the device attempting to enter to the network. Here, $\wedge$ represents the bitwise AND between two Bloom filters and $||$ denotes the number of 1s of the resulting AND. Equation 4 calculates the fraction of 1s in common between a network host bad model and a new device bad model with respect to the number of ones in the network host's

bad model. In other words, the cardinality of the AND measures how different or similar two models are to each other. If the device's bad model is equal to or it is a superset of server i bad model, its final vote is $v_i = 1$. Otherwise, if the new device's bad model is a subset of the server i bad model its final vote is $v_i = 0$. The final group vote is represented by the percentage of devices that agree on the decision, as expressed in equation 1.

In order to avoid attacks in which the new device presents a Bloom filter filled with all 1s as its bad model, or one computed with good and bad traffic meant to trick the vote in equation 4, the presented bad model is also checked against all normal profiles already in the network. If normal traffic is detected as part of the bad model, the device is rejected. This process is accomplished by calculating the AND cardinality of the presented bad model with each of the host profiles. Any cardinality above the false positive rate of the Bloom filter (which means that common n-grams exist) will reject the device.

Table 1. Voting process results for the Pre-connect phase. *ACC* denotes a device accepted to the network and *REJ* denotes a device rejected from entering to the network.

Scenario	server4_in	server3_in	server2_in	server1_in
(i) 33%	REJ	ACC	ACC	ACC
(ii)66%	REJ	REJ	ACC	ACC
(iii) 100%	REJ	REJ	REJ	ACC

Table 1 shows the pre-connect results after conducting the voting process for the three different scenarios previously described. The top entry in each column represents the server attempting to get connected to the network. The remaining three servers represent the devices making the decision. For instance, in Column 2 we assume that the hosts already in the network are *server1*, *server2* and *server3* while *server4* is attempting to enter the network. In Column 3, *server1*, *server2* and *server4* are considered to be the network hosts and *server3* is the one attempting to access the network. Similar reasonings apply to the remaining columns.

As can be seen in Column 2, *server4* is rejected in all cases since its bad model is the one with the least malware knowledge of all. When *server3* attempts to enter the network, its malware knowledge is a superset of *server4*. Therefore, it is only accepted when one device needs to agree on the pre-connect decision. However, because *server3* is only a subset of the bad models of *server2* and *server1*, it is not accepted for higher rates of required agreement. Similar reasonings apply for *server2* and *server1*. Note that different specific percentages in the voting process result in a more or less strict access control. More importantly, the results show that as long as the specific percentage of clean profiles is kept in the network, the voting process will be resilient to attacks by malicious devices that lied about its bad profile in order to manipulate access control. Future work will focus on applying control-theoretic concepts in a feed-back loop to provide an automated means of calibrating the sensitivity of the decision process. In case of failure, a network manager may fine tune the decision process to impose a predefined policy.

Devices rejected during this phase are placed in quarantine where the device's bad model is tested against a group of known attacks, so that a new reinforced bad model (Bloom filter) can be computed. The main advantage of having the malware knowledge in a model (Bloom filter), as opposed to having a list of signatures, is the fast processing time. AND-ing Bloom filters and calculating cardinalities is a much faster process that comparing signatures. Finally, we emphasize the importance of balancing the strictness of the access control with the latency of the system. Obviously, very demanding access control policies typically result in longer latencies due to quarantines. However, less demanding access control policies risk further attacks to the network.

4.2 Evaluation of the Post-connect Phase

We used the third week of collected traffic to test the post-connect phase in the BB-NAC architecture. For every incoming packet to any of the webservers, each server votes on the normalcy of the packet using its model or behavior profile. The evaluation of the post-connect phase is achieved by computing the false positive (FP) and detection rates (DR) of the voting process. In this context, FP represents the percentage of normal traffic falsely identified as anomalous by the group of network hosts, while DR denotes the percentage of bad traffic deemed as anomalous by the group of hosts. In order to measure the FP and DR of the voting process, we poisoned the collected traffic with the following known worms and viruses captured from real traffic: three versions of CodeRed, CodeRed II, WebDAV, a php forum attack, Mirela and the nsiislog.dll buffer overflow vulnerability (MS03-022) which exploits the IIS Windows media service.

We explore four different scenarios for the voting process: (i) A 25% of agreement among webservers is required. Since we are only considering four webservers this translates to a voting process in which only one vote is needed for an anomalous designation. (ii) A 50% of agreement among webservers is required, which means that at least two webservers have to agree on the anomalous nature of the observed traffic. (iii) 75% of the webserver's profiles have to agree on the decision (3 webservers in our network) and (iv) A 100% agreement, in which all profiles have to agree on the decision. These percentages represent quite a large spread designed to reveal the trend of the FP and DR in the voting process. Because we are using the content-based sensor Anagram as the AD, each profile votes based on whether the content of the packet (n-grams) being tested was seen during the training of the AD.

Table 2 summarizes the group FP and DR rates for the four scenarios described. As can be seen, when only one server vote is sufficient to decide whether the traffic is anomalous (25% row), the DR is 100% and the FP is 0.032%. As the percentage of servers that need to agree increases, the DR decreases since it becomes more difficult

Table 2. DR and FP: Group rates

Percentage	DR	FP
(i) 25%	100%	0.032%
(ii) 50%	99%	0.02%
(iii) 75%	99%	0.005%
(iv) 100%	83%	0.001%

Table 3. DR and FP: Individual rates

Server	DR	FP
server1	100%	0.02%
server2	83%	0.009%
server3	99%	0.015%
server4	99%	0.01%

for the four profiles to agree on the identification of anomalous traffic. On the other hand, the FP rate decreases considerably as the percentage of servers that have to agree increases. Obviously, with more servers involved in the vote there is a greater amount of information about normal traffic and hence it is less probable for normal packets to be mistakenly classified as anomalous. Given that high DR and low FP are the objectives of a good sensor, it appears that choosing an agreement of the 75% of the servers provides the *best collaborative solution* for the architecture. Such policy guarantees a very low FP of 0.005% and a DR of 99%. Other percentages translate into either smaller DR or larger FP rates. As in the pre-connect phase, the results also show that as long as the specific percentage of clean profiles is met (e.g., 75% in our example), the voting process will be robust to attacks by groups of malicious profiles trying to manipulate the vote. For example, in a network with 100 initial clean profiles, a group attack would need to introduce at least 35 malicious profiles in order to dilute the 75% agreement (75% of 135 profiles is 101 and the network would only have 100 clean profiles).

To test our theory that collaborating ADs are more powerful than individual ADs, we ran an experiment where only the server-specific AD's tests a packet without taking into account the decision of other devices with similar behavior. In this setting, only the server that is the destination of the traffic votes on the normalcy of the packets. As in the previous experiment, the third week of collected traffic was used together with real worms to poison the traffic. Table 3 shows the FP and DR for each of the servers when they run their own individual AD. The main conclusion drawn from comparing the *best collaborative solution* in Table 2 with Table 3 is that groups of ADs collaborating on the decision of normalcy or anomalous nature of the traffic, typically enhance the FP, the DR or both global rates when compared to individual ADs. For instance, in the case of *server2*, its individual DR is 83% and its individual FP rate is 0.009%. In contrast, the *best collaborative solution* results in an improved DR of 99% and a lower FP rate of 0.005%. While one may argue that in some instances the individual DR improves and the FP rate worsens (e.g., *server1*), the sum of all the individual ADs will always be worse off than the *best collaborative solution*. We conclude that the collaborative voting process improves the security enforcement of our architecture.

4.3 NAC Security Enforcement over Time: Concept Drift

We present a preliminary analysis on how our architecture conforms to *concept drift* i.e., the automatic update of security enforcement policies over time. The motivation is to account for and distinguish changes in the normal behavior of users from changes in behavior generated by an attack. Previous works such as FLORA [12] and STAND [2] considered algorithms in which the sensor only trusted the latest observed samples. In both, new samples were added to a set as they arrived, subsequently deleting the old samples. Furthermore, STAND detected anomalous behavior by comparing continuous models over time. We borrow these ideas in order to show how the voting process implemented in BB-NAC conforms to concept drift. As it is structured, Anagram considers a model stable whenever the amount of new, unseen n-grams is below a certain threshold [11]. If we assume a first approach where new models are computed keeping the information from previous models, the DR rate will eventually start to decrease. This corresponds to the expectation that the longer the training period goes, the more

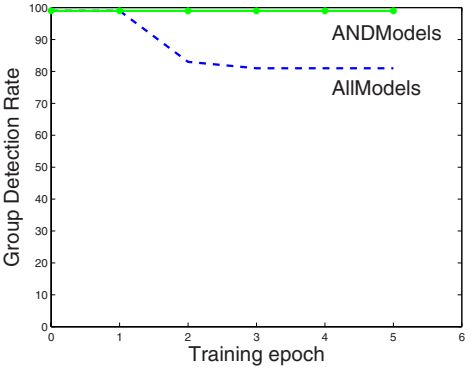


Fig. 4. Group Detection Rate for two alternative concept drift algorithms: *ANDModels* and *AllModels*

likely that bad data will be used in the modeling. To show this directly, we collected traffic in the Computer Science department at Columbia University for a period of two weeks. Using Anagram, new models were computed until they reached stability while *keeping all the information in the Bloom filter from the previous models*. We refer to this technique as *AllModels*. In Figure 4 we plot the DR for a group of four webservers with the *best collaborative solution* i.e., a packet is considered anomalous if 75% of the servers agree. Units in the x axis represent each moment in time (epoch) when one or more servers in the network computed a new behavior profile. Initially, the group DR for the four webservers starts at 99%. Next, *server1* computes a new model (epoch 1) and the group DR remains constant. We then proceed to poison the training traffic for *server1* and *server2* and have both compute new models (epoch 2). As a result, at epoch 2 the group detection rate decreases to 84% due to the fact that two profiles are poisoned and thus fail to correctly classify the traffic. Subsequent epochs involve the computation of new models by all the servers. However, the group anomaly detection is permanently damaged and does not vary from DR of 84%. The explanation behind this damage is that the training process just adds n-grams to the previous old models but still keeps the content of the attacks. Therefore, we conclude that a different approach is needed.

An alternative approach is one where we start new clean models every time a model is trained. In such a case, the profiles erase previously seen information that may be repeated in the future. A direct consequence of this approach is an expensive increase of the FP rate. Thus, the elimination of previously seen information does not appear to be a viable alternative. Instead, we opt for a solution where every time a new Bloom filter profile is computed, the new profile is AND-ed with all its previous q profiles keeping only common data seen in continuous training sets such that: $P_i = P_i \wedge P_{i-1} \ldots \wedge P_{i-q}$. We refer to this technique as *ANDModels*. This process allows us to detect and eliminate anomalous content that may have poisoned the models while they were being trained as shown by [2]. By keeping only the common data observed in q consecutive models, we guarantee that as long as we have one initial clean model, future models will also be clean. Obviously, the moment a device acquires a new different behavior, the

resulting *ANDModel* may be almost empty since old and new profiles might not have much content in common. This situation would generate a very high FP rate because the Bloom filter contains very few *normal* n-grams. As a solution, we repeat the AND process s times and OR the results as shown in Equation 5. Each *ANDModel* represents a clean model in the past, and by OR-ing them we stitch together the last s old and new behaviors [12].

$$P_i = OR_{(t=0,s-1)} \ (P_{i-t} \ \wedge \ P_{i-1-t} \ ... \ \wedge \ P_{i-q-t}) \tag{5}$$

The higher the value of q in Equation 5, the more difficult it will be for the enemy to attack the architecture since all q models would have to be poisoned. Similarly, the higher the value of s, the more *old* behaviors are kept in the model. Going back to our previous example in Figure 4 where *server1* and *server2* had permanently damaged the DR of the architecture, we applied *ANDModels* with values q=2, s=1 and repeated the simulation (Figure 4). With the new algorithm, the DR retains its initial value along the various epochs. In our experiments, we also note an increase in the FP rate of the sensor, probably due to the fact that *ANDModels* is eliminating content seen only in the last training period. The approach introduced here demonstrates that our architecture conforms to concept drift.

4.4 BB-NAC Latency Analysis

We estimate the latency of the pre-connect phase as follows:

$$l = l_a + (1 - \rho) \times l_q \tag{6}$$

where l_a represents the latency of the AND-ing between Bloom filters, variable ρ represents the probability that a device has an up-to-date bad model, and l_q represents the latency of the quarantine. For every AND operation, we estimated $l_a \approx 18$ms [1]. In case the AND operations cannot be run in parallel, l_a should be multiplied by the number of hosts in the cluster. For devices with up-to-date bad models, $\rho = 1$ and Equation 6 becomes $l = l_a$. On the other hand, if a device does not have an up-to-date bad model, it is quarantined and provided with a new bad model that represents the bad knowledge from all the other hosts. This new bad model is computed by OR-ing the host's bad models, where each OR operation is completed in approximately 18ms. As an example, l ranges from 180ms to 342ms for a cluster of 10 devices and $\rho = 0$.

The latency per packet during the post-connect phase is calculated as presented by Wang et al. in [11]:

$$l = ((1 - FP) \times l_{BF}) + (FP \times l_q) \tag{7}$$

where l_{BF} is the latency to check whether a certain n-gram is found in the profile's Bloom filter, and FP stands for false positive rate. A typical value for l_{BF} corresponds to about 5ms. If the checks cannot be performed in parallel for all profiles, l_{BF} would translate to $n \times l_{BF}$ where n stands for the number of hosts in the cluster responsible for the access control decision. For a cluster of 10 devices and a $FP = 0.005$, $l \approx$ 5.785ms–50.56ms.

[1] The numerical values discussed in this subsection were obtained using a 1.73GHz *Intel Pentium M Processor* and a set of Bloom filters of size 16MB.

5 Conclusions and Future Work

In this paper, we have introduced a novel NAC architecture, BB-NAC, which enforces security based on the exchange of behavior profiles. Each host in the network is represented with a profile and a bad model which are then used during pre-connect and post-connect phases to detect up-to-date malware knowledge and zero-day attacks. Our architecture enhances previous NAC technologies by automatically updating the behavior-based security policies according to the hosts' behavior evolution on a per-port basis. The experiments serve as a proof-of-concept for the novel behavior-based network access control presented here. We have shown that ADs collaborating through a voting process offer a more powerful approach to enforce security over individual ADs. Furthermore, our experiments confirm that BB-NAC is resilient to attacks even after accepting a percentage of malicious hosts into the network.

Future work will include evaluating the two-tier strategy for additional ports. We are also investigating the clustering of devices based on their profiles instead of a *self declaration of nature*. Lastly, we plan to evaluate the performance of BB-NAC when using non-content anomaly sensors. For this purpose, we are currently designing a non-content AD that we plan to use in order to reproduce similar pre-connect and post-connect tests for the architecture.

Acknowledgements

This work was partially supported by NSF Grant CNS-06-27473 and by DARPA Grant HR0011-06-1-0034. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF or the U.S. Government.

References

[1] Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. Communications of the ACM 13(7) (July 1970)
[2] Cretu, G., Stavrou, A., Stolfo, S., Keromytis, A.: Data sanitization: Improving the forensic utility of anomaly detection systems. In: Proceedings of the Third Workshop on Hot Topics in System Dependability (2007)
[3] Dressler, F., Munz, G., Carle, G.: Attack detection using cooperating autonomous detections systems (cats). In: Wilhelm-Schickard Institute of Computer Science, Computer Networks and Internet (2004)
[4] Necula, G.C.: Proof-carrying code. In: The 24th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 1997 (1997)
[5] Necula, G.C., Lee, P.: Safe kernel extensions without run-time checking. In: 2nd Symposium on Operating Systems Design and Implementation, OSDI 1996 (October 1996)
[6] Necula, G.C., Lee, P.: Efficient representation and validation of proofs. In: IEEE Symposiym on Logic in Computer Science, LICS 1998 (1998)
[7] Papadopoulos, C., Lindell, R., Mehringer, J., Hussain, A., Govindan, R.: Cossack: Coordinated suppression of simulatenous attacks. In: Proceedings of DISCEX III (2003)
[8] Parekh, J., Wang, K., Stolfo, S.: Privacy-preserving payload-based correlation for accurate malicious traffic detection. In: Large Scale Attack Defense, LSAD (2006)

[9] Snort rulesets, http://www.snort.org/pub-in/downloads.cgi
[10] VXHeavens, vx.netlux.org
[11] Wang, K., Parekh, J., Stolfo, S.: Anagram: A content anomaly detector resistant to mimicry attack. In: Zamboni, D., Krügel, C. (eds.) RAID 2006. LNCS, vol. 4219. Springer, Heidelberg (2006)
[12] Widmer, G., Kubat, M.: Learning in the presence of concept drift and hidden contexts. Machine Learning 23(1), 69–101 (1996)

Path-Based Access Control for Enterprise Networks*

Matthew Burnside and Angelos D. Keromytis

Computer Science Department
Columbia University
{mb,angelos}@cs.columbia.edu

Abstract. Enterprise networks are ubiquitious and increasingly complex. The mechanisms for defining security policies in these networks have not kept up with the advancements in networking technology. In most cases, system administrators define policies on a per-application basis, and subsequently, these policies do not interact. For example, there is no mechanism that allows a web server to communicate decisions based on its ruleset to a firewall in front of it, even though decisions being made at the web server may be relevant to decisions at the firewall. In this paper, we describe a path-based access control system for service-oriented architecture (SOA)-style networks which allows services to pass access-control-related information to neighboring services, as the services process requests from outsiders and from each other. Path-based access control defends networks against a class of attacks wherein individual services make correct access control decisions but the resulting global network behavior is incorrect. We demonstrate the system in two forms, using graph-based policies and by leveraging the KeyNote trust management system.

Keywords: Path-based, access control, Keynote, SOA, enterprise.

1 Introduction

Most enterprise networks are distributed structures with multiple administrative domains and heterogeneous components. Defining and enforcing security policies in these networks is challenging – it is difficult for a system administrator or group of system administrators to conceptualize the security policy for such a network, let alone correctly express that policy in the myriad of languages and formats required by such an environment.

Consider a system where an oracle responds to all policy requests from the network. The complete, high-level policy for the network, as defined by the system

* This work was partially supported by NSF Grant CNS-07-14647 and by ONR MURI Grant N00014-07-1-0907. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF or the U.S. Government.

T.-C. Wu et al. (Eds.): ISC 2008, LNCS 5222, pp. 191–203, 2008.

administrator, is stored in and evaluated at the oracle. For every policy decision, a service queries the oracle and acts based on its response. Such a system provides a globally coherent policy, but clearly does not scale well. Therefore, it is common practice to derive from the system administrators' high-level conceptual policy a set of policy components where each component is deployed at a single service or node. Each policy component is translated into the appropriate language for the target service and deployed directly at that service. In most cases, this task is performed by hand by the system administrator, though there have been some attempts at automating it, as in [1] [2].

1.1 Example

Fundamentally, there is a violation of assumptions that comes from taking a high-level conceptual policy and componentizing it, either manually or mechanically. Consider the simple e-commerce network in Fig. 1. A firewall protects several hosts. On the first host are a web server and some business logic in the form of, *e.g.*, PHP or ColdFusion; on the second host is a database.

We propose a high-level conceptual policy for this network: all connections should arrive at port 80 on the firewall, authenticate at web server with a username and password, and the business logic must authenticate to the database using a public-key pair.

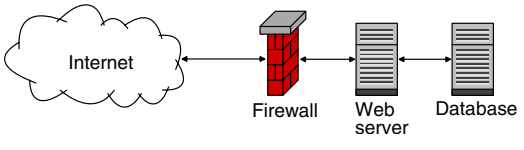


Fig. 1. A simple network. A web server and database are connected to the Internet through a firewall.

The system administrator derive from that high-level conceptual policy a set of policy components. One policy component is the firewall ruleset which blocks traffic to all ports except TCP port 80. Another component is the `.htaccess` file on the web server indicating that only a set of username/passwords may access the files containing the business logic. A final component is the grant table at the database which indicates that only the key pair used by the business logic may access the tables for that application. It is our contention that, in the process of generating these policy components, *path* information has been lost.

Consider an unknowing or malicious employee who plugs in a wireless access point, as in Fig. 2. An adversary can connect to this wireless access point and, through it, probe the web server and database. Such a connection violates the conceptual policy determined by the system administrator, but none of the individual policy mechanisms in place will detect it. That is, none of the policy mechanisms (the firewall ruleset, the `.htaccess` file, *etc.*)allows for governing how a request arrived at the service, but only what it requests after arrival.

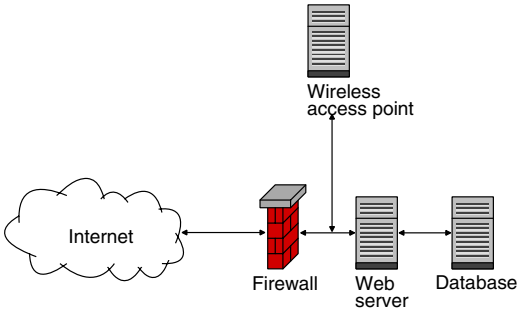


Fig. 2. A vulnerable network

Similar flaws may occur if, for example, the firewall accidentally fails open due to misconfiguration or routing changes, or if an adversary attempts to access the business logic through some routing path that was unintentionally enabled. A compromised internal machine may be used to probe the remainder of the network. A misconfigured router may allow connections to bypass a firewall.

1.2 Contributions

In this work, we dynamically model the paths that requests take as they traverse an enterprise network and use those models as the basis for informing policy decisions. Requests traversing invalid paths are barred from penetrating deeper into the network. In a service-oriented architecture (SOA), the type of network we focus on in this paper, the path of interaction followed by a request is a tree. The root of the tree is the first point of interaction with the network (the firewall in the example above) and the branches of the tree represent the various actions taken by the various services in the network in response to that request. Enforcing policy consists of examining each pathway to determine that it followed a proscribed route.

We use a binary view in the policy-enforcement mechanism. Either a policy-proscribed service is in a pathway, or it is not. However, we also collect additional fine-grain details about events in each pathway in order to perform aggregate analysis. By exposing more information to downstream services, it is possible for the policy engine at each service to make decisions based on historical data or statistical trends. Note that the statistical analysis is not the focus of this paper and we do not address it further.

Accumulating path-traversal information benefits an enterprise by providing a simple, low cost, mechanism for preventing attacks that violate system administrators' assumptions about allowed or valid pathways. For example, a rogue wireless point is no longer the danger it once was. A misconfigured firewall will be more easily detected and many attacks during the window of vulnerability will be prevented.

In this paper, we present two solutions. The first is a low-cost, high-performance system that models incoming requests as graphs, where events are

vertices and dependencies are edges. The second extends this concept and provides protection in some situations where internal nodes are untrusted; it leverages the KeyNote trust management system[3] [4]. We show that in both cases, the performance overhead on the SOA is low.

The remainder of this paper is organized as follows. In Sect. 2, we discuss related work in the field. In Sect. 3, we describe the architecture of our two solutions. In Sect. 4, we give details on their implementation. We evaluate the work in Sect. 5 and conclude in Sect. 6.

2 Related Work

Most prior work in the policy field can be divided into three major categories: policy specification [3,5], resolving policy conflicts [6,7], and distributed enforcement [8,9].

In their work in the field of trust management, Blaze, *et al.*, [10,11,12] built PolicyMaker, a tool that takes a unified approach to describing policies and trust relationships in enterprise-scale networks by defining policies based on credentials. It is based on a policy engine that identifies whether some request r with credentials c complies with policy p. In PolicyMaker, policies are defined by programs evaluated at runtime. SPKI [13,14,15] is a similar mechanism that uses a formal language for expressing policies. However, in both cases, the focus is on trust management rather than policy correctness. Both systems can be used as components in facilitating path-based access control, but alone, they are insufficient.

When there are multiple policies or multiple users defining policy there is always the possibility of conflict. Cholvy, *et al.* [7] describe a method for resolving that inconsistency and show that the problem is exacerbated in large-scale networks. As with PolicyMaker and SPKI, this method may facilitate path-based access control but it does not provide the information transfer necessary for resolving violated system administrator assumptions.

The STRONGMAN trust management system [2] focuses on the problem of scaling the enforcement of security policies and resolving policy conflicts. In STRONGMAN, high-level, abstract security policies are automatically translated into smaller components for each service in the network. STRONGMAN features no provision for future interaction between components, a key feature of path-based access control.

Bonatti, *et al.*, [16] propose an algebra for composing heterogeneous security policies. This is useful in networks with multiple policies defined in multiple languages (*i.e.*, most networks today). However, this system requires that *all* policies and supporting information and credentials be available at a single decision point, *e.g.* an oracle as discussed previously.

Firewalls [17,18] are one of the most common and most well-known mechanisms for policy enforcement. The Firmato system [19] is a firewall management toolkit for large-scale networks. It provides a portable, unified policy language, independent of the firewall specifics. Firewall configuration files are generated

automatically from the unified global policy. Firmato is limited to packet filtering, and it does not provide for future interactions between components.

The Oasis architecture [20] takes a wider view and uses a role-based system where principals are issued names by services. A principal can only use a new service on the condition that it has already been issued a name from a specific other service. Oasis recognizes the need to coordinate the dependencies between services, but since credentials are limited to verifying membership in a group or role, it is necessary to tie policies closely to the groups to which they apply.

In [21], the authors use KeyNote to distribute firewall rulesets, allowing endpoint nodes to perform enforcement independently. The path-based access control mechanism can be viewed as an extension of the distributed firewall system, allowing each endpoint node to incorporate the path of the request into its policy evaluation. The path-based system can further be viewed as an instantiation of the virtual private services described in [22]. Each request is presented a view of the network (a "private service") that is customized, based on the path the request has taken up to that point.

3 Architecture

In this section, we describe two methods for implementing path-based access control. The systems differ primarily in the mechanism by which the policy is evaluated. In the first system, we model policies and incoming requests as graphs, and we evaluate the policies by comparing the graphs representing actual requests with the policy graphs. The second system extends the first by modeling incoming requests and policies as KeyNote assertion chains.

Both systems are designed for use in SOA-style networks, so policy definition consists of defining trees representing valid requests. The policy distributed to each service is a list representing the path from the root to that service in the policy graph. This technique is simple and can be performed quickly, making it a good fit for dynamic networks where request patterns change quickly.

In both systems, the threat we consider is one where an adversary is attempting to access the network through unauthorized pathways. That is, pathways which have not been explicitly allowed by the system administrator.

3.1 Graph-Based Access Control

The goal of this system is to forward information about access control-related events at each service to subsequent services. The accumulated information is used by a policy engine co-located with each service to detect pathway-violation attacks.

At each service, a small program called a *sensor* observes information regarding access-control events and forwards that data to downstream nodes. We packetize this data and call each packet an *event*. Each sensor is situated such that it can observe its target service and report on the access-control decisions made therein. Typically, sensors are quite simple. For example, the sensor for

the Apache web server parses the Apache log and error files for reports on authorization attempts. Each entry for an authorization attempt in the log files is an event. The details of the event are associated with it as a set of attribute key-value pairs and reported to downstream services.

Second-order sensors, called correlation sensors, use additional information reported by the sensors to correlate events on a hop-by-hop basis. For example the events generated by the Apache sensor are correlated with packets departing the firewall based on the time, the source port, the IP address, and the TCP sequence number. The complete data set received by a downstream node is a chain, linking the incoming request with the source principal and all intermediate hops in the network. Thus, the policy decision made at a given service can incorporate the additional information obtained from upstream nodes.

As a request propagates through a network, the associated events are forwarded along with it. The events are represented as vertices in a graph, and the correlation information generated by the correlation sensors is used to form edges between them. When a request arrives at a service, it is accompanied by a graph representing the history of its interaction with the network.

Reactive systems like this, as with most intrusion detection systems, depend on the inviolability of the sensor network. This requires particular attention be spent securing the sensors. In this paper, we do not address attacks wherein the sensor network itself is compromised, though we do note that the KeyNote-based system will alleviate some of those attacks. Sensors may be further protected by lifting them into a hypervisory role, or by isolating sensors and applications through virtual machines, as in [23].

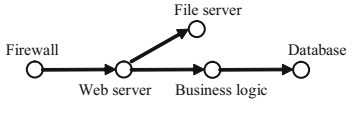


Fig. 3. The tree of applications handling a request

Note that the overall path taken by a request as it traverses a network is a tree, as in Fig. 3. However, the path taken by a request from its arrival in the network to a given node is a tree-traversal from the root to a leaf or internal node. This path is necessarily linear. As a request passes through a network, the events generated by the sensors associated with it represent the linear path of that request. By situating correlation sensors between hosts and between services, the graph is propagated across the network. Each node in the graph receives the access control decisions made by all its upstream nodes, and this is used to inform future access control decisions.

To enact the access control mechanism, we define the policy at each node as a graph, as shown in Fig. 4. The graph representing an incoming connection must match a policy graph in order for the connection to be accepted. However, since the graph is always linear, the policy takes the form of a list of services over which the request must traverse, and valid values for the key-value pairs

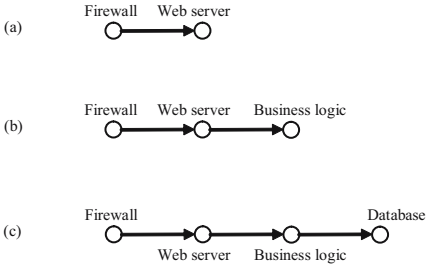


Fig. 4. A graphical representation of the policy at (a) the web server (b) the business logic and (c) the database

associated with each event. Any request taking an unexpected pathway or with non-matching attributes will necessarily be detected and rejected.

For example, the policy evaluation at the business logic consists of a traversal of the graph delivered from the upstream node to verify that each node from Fig. 4b appears, and is in the correct order.

3.2 KeyNote-Based Access Control

This method can be viewed as an extension of the graph-based access-control system. In the graph-based system, we build a path representing the route a request took from its entry in the network to a given host. In this system, we leverage the cryptographic tools and trust-management capabilities of the KeyNote system to instead build a certificate chain representing the path taken by a request from its entry in the network to a given host.

Like all reactive, sensor-based systems, the previously-described graph-based system is vulnerable to malicious internal nodes. That is, a compromised or otherwise malicious intermediate node on the path between an application and the entry point for a request can modify the graph dataset before forwarding it. The addition of the KeyNote system protects from some classes of such attacks.

In the KeyNote system, events are reported in the form of KeyNote credentials, and policy is evaluated at each service by a KeyNote compliance checker. Traditional KeyNote credentials allow principals to delegate authorization to other principals, while in the path-based access control scheme, KeyNote credentials delegate authorization for handling a request from a given application to the next downstream application.

When a request generates an event e_1 at host H_1 and the request is then forwarded then to host H_2 where it generates event e_2, a correlation sensor correlates the events as in the graph-based architecture. However, in this case the correlation notification takes the form of a KeyNote credential. That is, it is a signed assertion, with authorizer H_1 and licensee H_2, indicating that e_1 and e_2 are linked. For example, the following credential might be issued by a firewall when it redirects an incoming request to a web server.

```
KeyNote-Version: 2
Comment: Forward request to web server
Local-Constants:  FW_key = "RSA:acdfa1df1011bbac"
                  WEB_key = "RSA:deadbeefcafe001a"
Authorizer: FW_key
Licensees: WEB_key
Signature: "RSA-SHA1:f00f2244"
Conditions: ...
```

KeyNote provides an additional field `Conditions` which is used to encapsulate references to events e_1 and e_2. Credentials are chained such that the licensee for each event is designated as the next hop in the graph. In a simple e-commerce example, an event generated at a web server and passed to the database would include the previous credential along with the following.

```
KeyNote-Version: 2
  Comment: Send SQL SELECT statement to the DB
  Local-Constants: WEB_key = "RSA:deadbeefcafe001a"
                   DB_key = "RSA:101abbcc22330001"
  Authorizer: WEB_key
  Licensees: DB_key
  Signature: "RSA-SHA1:baba3232"
```

The first link of the credential chain is created by the firewall. This credential binds the principal (the TCP/IP address of the incoming connection) to the first hop in the chain. The key for the principal is randomly generated, and then cached, at the firewall. Such a credential takes the following form:

```
KeyNote-Version: 2
  Comment: New principal at the firewall
  Local-Constants: P_key = "RSA:ffeedd22eecc5555"
                   FW_key = "RSA:acdfa1df1011bbac"
  Authorizer: P_key
  Licensees: FW_key
  Conditions: hop0 == "PRINCIPAL"
  Signature: "RSA-SHA1:ceecd00d"
```

As a request progresses through the network, the result is a chain of credentials that link the incoming request at a given node back through each intermediate node to the principal.

The policy at each node is a list of keys, in order, that must be found in the credential chain. It is similar in concept to the policy definitions shown in Fig. 4, but with each node is also associated a key. As the set of credentials arrives at each node, the local KeyNote compliance checker verifies that the set comprises a chain. If successful, the policy engine then traverses the chain to verify that the keys occur in the order expressed in Fig. 4. If either step fails, the request is blocked.

4 Implementation

Each of these systems were implemented in the Python programming language. Sensors were written and deployed for the OpenBSD PF firewall, Apache, PHP, and MySQL, among other applications. These sensors parse the log files and observe authentication-related behavior of each application in order to generate events describing the access control behavior of each. The correlation sensor engine, an instance of which is deployed between each pair of neighboring sensors, maintains a cache of recently-observed events and generates correlation events based on runtime-configurable fields from the event descriptions. Each time a correlation between two events is made, the two events are linked and forwarded to the next-hop service, along with all previously accumulated events and correlations associated with the request.

At each service, requests are intercepted by a local firewall and redirected to the local policy engine. This engine delays the request until the associated graph arrives from the upstream node. The policy engine traverses the graph and verifies that it conforms to the administrator-defined policy. If the graph validates, the request is allowed to continue to the application, and the graph information is passed to the application sensor.

The KeyNote implementation is similar, but where the graph-based system generated events with arbitrary fields, this implementation generates KeyNote credentials using the KeyNote credential format. The policy for the credential chain is evaluated using the KeyNote compliance checker, through the `pykeynote` module.

5 Evaluation

We evaluated these two systems on a testbed network consisting of an OpenBSD PF firewall, an Apache web server running PHP 5.2.3, and a MySQL 5.0.45 server. The network is deployed as shown in Fig. 5. The only unblocked incoming port on the firewall is port 80. The firewall also performs network address translation (NAT) so the internal machines have IP addresses in the 10.0.0.0/24 netblock. The testbed application consists of a PHP application which loads and displays a 1MB image from the MySQL database.

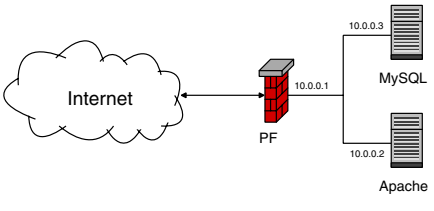


Fig. 5. Testbed network. The OpenBSD PF firewall protects an Apache web server and MySQL database.

The high-level conceptual policy for this network – that is, the policy as it might be expressed informally by the system administrator – is that all connections into this network must be vetted by the firewall to guarantee that they are arriving on the correct port, then processed by the web server and PHP engine, and finally passed to the database. In the attack scenario, a rogue wireless access point is attached to the network as shown in Fig. 6. This opens the potential for incoming connections to access the web server or database without first being processed by the upstream nodes – an *assumption*-violation attack. No events or path information will be generated by requests passing through the wireless access point, since, in this example, the system administrator is unaware of its existence and has not installed any sensors on it.

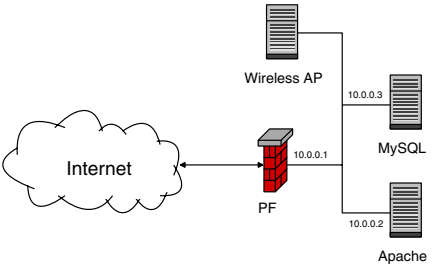


Fig. 6. Vulnerable testbed network. A wireless access point has been connected to the network, allowing traffic to the web server that has not traversed the firewall.

We evaluate the system on two fronts: performance and effectiveness. Performance is measured by timing batch requests made on the system. Effectiveness is analyzed by attempting to detect previously-unseen assumption-violating attacks.

The graph-based access control system is deployed on the testbed network as follows. Sensors are deployed on the network interfaces of all machines, including both network interfaces of the firewall, and at the firewall, web server, PHP engine, and database themselves. Correlation sensors are placed between each neighboring pair of nodes. When a request arrives from an external host, it is processed by the firewall and the sensor on the external network interface. As the request is subsequently processed by the firewall engine itself, and then forwarded out through the internal network interface, the sensors generate events which are linked by correlation sensors. The graph thus generated is collected and forwarded from node to node as the request progresses through the network.

The high-level policy for this network is that all requests must pass, in order, from the firewall to the web server to the database. We derive the actual policy for each node from the high-level policy by determining the path that a request must travel in order to reach that node. Thus, the policy at the database is that it will only handle requests which have traversed the firewall and web server. The policy at the web server is that it will only handle requests that have traversed the firewall. The policy definition for each service consists of an ordered list of nodes. Policy evaluation is a matter of traversing the linear graph built by the

sensors and correlation sensors to verify that the nodes occur and are in the correct order.

We test the effectiveness of this system by attempting to connect to the web server and database, through the wireless access point. Since the requests do not pass through the firewall, the graphs associated with the requests, do not have the firewall as the root node. The requests are therefore denied by the policy engine at the web server and database policy engines.

The KeyNote-based system is deployed on the same network. In KeyNote, the policy, rather than being a list of nodes, is a list of keys. The credential chain have have signed credentials, in the correct order, from each of those nodes. *E.g.*, the policy at the database is that the credential chain must have credentials signed by the web server and firewall, in that order. Policy evaluation consists of verification that the credential chain is, in fact a chain, and then a search of that chain for the policy key list.

One test of the effectiveness of the KeyNote system is similar to the tests for the graph-based system. Requests on the firewall are handled as expected, and requests through the wireless access point are blocked as the credential chains thus generated are incorrect.

We analyzing the performance of these systems by determining the overhead incurred by the additional network traffic and processing over the vanilla network. The test application deployed in this network loads files stored in a table in the MySQL database. The test file was 1 megabyte of binary data, and the time for the vanilla system to return that file, from request arrival to completion of the file transfer 162ms, averaged over 25 trials. The average handling time for the graph-based system was 317ms, averaged over 25 trials. The average handling time for the KeyNote-based system was 1.12s, averaged over 25 trials.

The majority of the overhead in the graph-based implementation is due to the delay in waiting for graph information to catch up to each request. The cost increase in the KeyNote-based system is due to the cryptographic costs inherent in the KeyNote system.

We find that in the graph-based system the overhead for a three-node network is 155ms, or approximately 50ms per node. In the KeyNote system, the overhead is 958ms, or approximately 320ms per node. As before, the additional overhead in the KeyNote-based system comes from the substantial cryptographic requirements of the KeyNote architecture.

6 Conclusion

In this work, we have described a mechanism for enhancing the current paradigm of access control to protect against a new class of attacks. These attacks take advantage of the fact that, in the process of converting a security policy from its conceptual, high-level, format to its distributed, low-level, form, information is lost. We describe two systems for defending against this new class of attacks, by passing path information from service to service as the request traverses the network. We show that the overhead incurred in these systems is relatively low.

References

1. Ioannidis, S.: Security policy consistency and distributed evaluation in heterogeneous environments. PhD thesis (2007)
2. Keromytis, A.D., Ioannidis, S., Greenwald, M.B., Smith, J.M.: The STRONGMAN Architecture. In: Proceedings of the 3^{rd} DARPA Information Survivability Conference and Exposition (DISCEX III), pp. 178–188 (April 2003)
3. Blaze, M., Feigenbaum, J., Ioannidis, J., Keromytis, A.D.: The KeyNote Trust Management System Version 2. Internet RFC 2704 (September 1999)
4. Blaze, M., Feigenbaum, J., Keromytis, A.: KeyNote: Trust Management for Public-Key Infrastructures. In: Christianson, B., Crispo, B., Harbison, W.S., Roe, M. (eds.) Security Protocols 1998. LNCS, vol. 1550, pp. 59–63. Springer, Heidelberg (1999)
5. Damianou, M.: A Policy Framework for Management of Distributed Systems. PhD thesis (2002)
6. Jajodia, S., Samarati, P., Subrahmanian, V.S.: A logical language for expressing authorizations. In: Proceedings of the 1997 IEEE Symposium on Security and Privacy, pp. 31–42 (May 1997)
7. Cholvy, L., Cuppens, F.: Analyzing consistency of security policies. In: RSP: 18th IEEE Computer Society Symposium on Research in Security and Privacy (1997)
8. Thompson, M., Johnston, W., Mudumbai, S., Hoo, G., Jackson, K., Essiari, A.: Certificate-based access control for widely distributed resources. In: Proceedings of the USENIX Security Symposium, pp. 215–228 (August 1999)
9. Keromytis, A.D., Ioannidis, S., Greenwald, M.B., Smith, J.M.: Managing access control in large scale heterogeneous networks. In: Proceedings of the NATO NC3A Symposium on Interoperable Networks for Secure Communications (INSC) (November 2003)
10. Blaze, M., Feigenbaum, J., Lacy, J.: Decentralized Trust Management. In: Proc. of the 17th Symposium on Security and Privacy, pp. 164–173. IEEE Computer Society Press, Los Alamitos (1996)
11. Blaze, M., Feigenbaum, J., Strauss, M.: Compliance Checking in the PolicyMaker Trust-Management System. In: Hirschfeld, R. (ed.) FC 1998. LNCS, vol. 1465, pp. 254–274. Springer, Heidelberg (1998)
12. Blaze, M., Feigenbaum, J., Ioannidis, J., Keromytis, A.: The role of trust management in distributed systems security. In: Secure Internet Programming, pp. 185–210.
13. Ellison, C.: SPKI requirements. Request for Comments 2692, Internet Engineering Task Force (September 1999)
14. Ellison, C., Frantz, B., Lampson, B., Rivest, R., Thomas, B., Ylonen, T.: SPKI certificate theory. Request for Comments 2693, Internet Engineering Task Force (September 1999)
15. Ellison, C.M.: SDSI/SPKI BNF. Private Email (July 1997)
16. Bonatti, P., di Vimercati, S.D.C., Samarati, P.: A Modular Approach to Composing Access Policies. In: Proceedings of Computer and Communications Security (CCS 2000), pp. 164–173 (November 2000)
17. Cheswick, W.R., Bellovin, S.M.: Firewalls and Internet Security: Repelling the Wily Hacker. Addison-Wesley, Reading (1994)
18. Mogul, J., Rashid, R., Accetta, M.: The Packet Filter: An Efficient Mechanism for User-level Network Code. In: Proceedings of the Eleventh ACM Symposium on Operating Systems Principles, pp. 39–51 (November 1987)

19. Bartal, Y., Mayer, A., Nissim, K., Wool, A.: Firmato: a novel firewall management toolkit. In: Proceedings of the 1999 IEEE Symposium on Security and Privacy, pp. 17–31 (May 1999)
20. Hayton, R., Bacon, J., Moody, K.: Access Control in an Open Distributed Environment. In: IEEE Symposium on Security and Privacy (May 1998)
21. Ioannidis, S., Keromytis, A.D., Bellovin, S.M., Smith, J.M.: Implementing a distributed firewall. In: 7th ACM International Conference on Computer and Communications Security (CCS), pp. 190–199 (November 2000)
22. Ioannidis, S., Bellovin, S.M., Ioannidis, J., Keromytis, A.D., Anagnostakis, K.G., Smith, J.M.: Virtual private services: Coordinated policy enforcement for distributed applications. International Journal of Network Security (IJNS) 4(1), 69–80 (2007)
23. Dunlap, G.W., King, S.T., Cinar, S., Basrai, M.A., Chen, P.M.: Revirt: enabling intrusion analysis through virtual-machine logging and replay. In: OSDI 2002: Proceedings of the 5th Symposium on Operating Systems Design and Implementation, pp. 211–224. ACM, New York (2002)

Cryptanalysis of Rabbit

Yi Lu[1], Huaxiong Wang[1,2], and San Ling[1]

[1] Division of Mathematical Sciences
School of Physical & Mathematical Sciences
Nanyang Technological University, Singapore
[2] Centre for Advanced Computing - Algorithms and Cryptography
Department of Computing
Macquarie University, Australia
luyi666@gmail.com, hxwang@ntu.edu.sg, lingsan@ntu.edu.sg

Abstract. The stream cipher Rabbit is one candidate to the ECRYPT Stream Cipher Project (eSTREAM) on the third evaluation phase. It has a 128-bit key, 64-bit IV and 513-bit internal state. Currently, only one paper [1] studied it besides a series of white papers by the authors of Rabbit. In [1], the bias of the keystream sub-blocks was studied and a distinguishing attack with the estimated complexity 2^{247} was proposed based on the largest bias computed.

In this paper, we first computed the exact bias of the keystream sub-blocks by Fast Fourier Transform (FFT). Our result leads to the best distinguishing attack with the complexity 2^{158} so far, in comparison to 2^{247} in [1]. Meanwhile, our result also indicates that the approximation assumption used in [1] is *critical* for estimation of the bias and cannot be ignored. Secondly, our distinguishing attack is extended to a multi-frame key-recovery attack, assuming that the relation between part of the internal states of all frames is known. Our attack uses $2^{51.5}$ frames and the first three keystream blocks of each frame. It takes memory $O(2^{32})$, precomputation $O(2^{32})$ and time $O(2^{97.5})$ to recover the keys for all frames. This is the first known key-recovery attack on Rabbit, though the attack assumption is unusually strong. Lastly, as an independent result, we introduced the property of Almost-Right-Distributivity of the bit-wise rotation over the modular addition for our algebraic analysis. This allows to solve the nonlinear yet symmetric equation system more efficiently for our problem.

Keywords: Rabbit, ECRYPT, stream cipher, keystream, bias, distinguishing attack.

1 Introduction

Rabbit is a stream cipher, which was first presented at [4]. It is one candidate [5] to the ECRYPT Stream Cipher Project (eSTREAM), which is a multi-year effort to identify new stream ciphers that might become suitable for widespread adoption. Since ECRYPT launched its call for primitives in the late 2004, it

T.-C. Wu et al. (Eds.): ISC 2008, LNCS 5222, pp. 204–214, 2008.

received 34 primitives in total. Now, eSTREAM is on the third evaluation phase. It currently has 16 remaining candidates, including Rabbit.

Rabbit was designed for both hardware and software implementations. It has a 128-bit key (also supports the 80-bit key), 64-bit IV and 513-bit internal state. Besides the reference paper [5], the authors publish a series of white papers [7,8,9,10,11,12] in support of its resistance to the well-known attacks (e.g. algebraic attack, correlation attack, guess-and-determine attack, differential attack). In academia, Rabbit is still considered to be a strong cipher. So far, only one paper [1] studied it and the work was on the bias. In [1], the core function g was shown to be unbalanced. The largest bias in the keystream was estimated to be $2^{-123.5}$ in [1]. It leads to a distinguishing attack, which requires 2^{247} keystream sub-blocks generated from random keys and IV's. This is the only known distinguishing attack on Rabbit, though the complexity is higher than the exhaustive search 2^{128}.

The main contribution in this paper is the following. First, we computed the exact bias of the keystream sub-blocks produced by Rabbit by Fast Fourier Transform (FFT). Our computation is based on the fact (cf. [14]) that the distribution of the sum of three addends can be computed from FFT of the individual distribution of each addend. Our result leads to the best distinguishing attack with the complexity 2^{158} so far, compared with the estimated complexity 2^{247} in [1]. It's much closer to yet still higher than the exhaustive search complexity 2^{128}. Our result also indicates that the approximation assumption used in [1] is *critical* for estimation of the bias and cannot be ignored. Secondly, assuming that we know the relation between part of the internal states of all frames, we extend our distinguishing attack to a multi-frame key-recovery attack. Our attack uses $2^{51.5}$ frames and the first three keystream blocks of each frame. It takes memory $O(2^{32})$, precomputation $O(2^{32})$ and time $O(2^{97.5})$ to recover the keys for all frames. This is the first known key-recovery attack on Rabbit, though the attack assumption is unusually strong. Lastly, as an independent result, we introduced the property of Almost-Right-Distributivity of the bit-wise rotation over the modular addition for our algebraic analysis. This allows us to solve the nonlinear yet symmetric equation system for our problem in time 2^{46}, which is more efficient than the naive guess-and-determine method taking time 2^{64}.

The rest of the paper is organized as follows. In Section 2, we describe the stream cipher Rabbit. We then analyze the bias of Rabbit and study the distinguishing attack in Section 3. We explore the possibility of extending our distinguishing attack to a multi-frame key-recovery attack in Section 4. Finally, we conclude in Section 5.

2 Description of Rabbit

Rabbit has a key size of 128 bits and an internal state of 513 bits. The internal state of Rabbit at time t ($t \geq 1$) includes 16 words of 32 bits $x_{0,t}, \ldots, x_{7,t}$, $c_{0,t}, \ldots, c_{7,t}$ and one-bit $\phi_{7,t}$. In total the internal state has $16 \times 32 + 1 = 513$ bits. They are updated at each clock as follows.

$$c_{j,t} = \begin{cases} c_{0,t-1} + a_0 + \phi_{7,t-1} \mod 2^{32}, & \text{if } j = 0 \\ c_{j,t-1} + a_j + \phi_{j-1,t} \mod 2^{32}, & \text{if } j > 0 \end{cases}$$

where

$$\phi_{j,t+1} = \begin{cases} 1, & \text{if } j = 0 \text{ and } c_{0,t} + a_0 + \phi_{7,t} \geq 2^{32} \\ 1, & \text{if } j > 0 \text{ and } c_{j,t} + a_j + \phi_{j-1,t+1} \geq 2^{32} \\ 0, & \text{otherwise.} \end{cases}$$

The a_i's are constants in hexadecimal representation: $a_0 = a_3 = a_6 = 0x4D34D34D$, $a_1 = a_4 = a_7 = 0xD34D34D3$, $a_2 = a_5 = 0x34D34D34$.

$$x_{0,t+1} = g_{0,t} + (g_{7,t} \lll 16) + (g_{6,t} \lll 16) \tag{1}$$
$$x_{1,t+1} = g_{1,t} + (g_{0,t} \lll 8) + g_{7,t} \tag{2}$$
$$x_{2,t+1} = g_{2,t} + (g_{1,t} \lll 16) + (g_{0,t} \lll 16) \tag{3}$$
$$x_{3,t+1} = g_{3,t} + (g_{2,t} \lll 8) + g_{1,t} \tag{4}$$
$$x_{4,t+1} = g_{4,t} + (g_{3,t} \lll 16) + (g_{2,t} \lll 16) \tag{5}$$
$$x_{5,t+1} = g_{5,t} + (g_{4,t} \lll 8) + g_{3,t} \tag{6}$$
$$x_{6,t+1} = g_{6,t} + (g_{5,t} \lll 16) + (g_{4,t} \lll 16) \tag{7}$$
$$x_{7,t+1} = g_{7,t} + (g_{6,t} \lll 8) + g_{5,t} \tag{8}$$

where $\lll$ denotes left bit-wise rotation, all additions (denoted by $+$) are computed modulo 2^{32} and $g_{j,t}$ for $j = 0, \ldots, 7$ is computed from $x_{j,t}$ and $c_{j,t+1}$. It's defined by

$$g_{j,t} = (x_{j,t} + c_{j,t+1})^2 \oplus \left((x_{j,t} + c_{j,t+1})^2 \gg 32\right). \tag{9}$$

Here, the additions are computed modulo 2^{32} and the squares are computed over integers and represented by 64 bits. In other words, $g_{j,t}$ just computes the bit-wise XOR of the higher 32-bit half of the square with the lower 32-bit half of the square.

2.1 Initialization

For simplicity, in this paper we concentrate on initialization by the key only according to the paper [4]. Let the 128-bit key $K = k_7 k_6 \cdots k_0$, where each k_i has 16 bits. We set the internal state at time $t = -4$ as follows:

$$x_{j,-4} = \begin{cases} k_{(j+1 \mod 8)} \| k_j, & \text{for even } j \\ k_{(j+5 \mod 8)} \| k_{(j+4 \mod 8)}, & \text{for odd } j \end{cases}$$

and

$$c_{j,-4} = \begin{cases} k_{(j+4 \mod 8)} \| k_{(j+5 \mod 8)}, & \text{for even } j \\ k_j \| k_{(j+1 \mod 8)}, & \text{for odd } j \end{cases} \tag{10}$$

Let $\phi_{7,-4} = 0$. Then the internal state is updated as usual for four clocks. After that, we modify each $c_{j,0}$ by $c_{j,0} = c_{j,0} \oplus x_{(j+4 \mod 8),0}$. From the next clock $t = 1$, the internal state is updated regularly and produces the keystream output as follows.

2.2 Keystream Generation

At each clock $t \geq 1$, Rabbit produces a 128-bit keystream block s_t by

$$\begin{aligned} s_t^{[15..0]} &= x_{0,t}^{[15..0]} \oplus x_{5,t}^{[31..16]} & \qquad s_t^{[31..16]} &= x_{0,t}^{[31..16]} \oplus x_{3,t}^{[15..0]} \\ s_t^{[47..32]} &= x_{2,t}^{[15..0]} \oplus x_{7,t}^{[31..16]} & \qquad s_t^{[63..48]} &= x_{2,t}^{[31..16]} \oplus x_{5,t}^{[15..0]} \\ s_t^{[79..64]} &= x_{4,t}^{[15..0]} \oplus x_{1,t}^{[31..16]} & \qquad s_t^{[95..80]} &= x_{4,t}^{[31..16]} \oplus x_{7,t}^{[15..0]} \\ s_t^{[111..96]} &= x_{6,t}^{[15..0]} \oplus x_{3,t}^{[31..16]} & \qquad s_t^{[127..112]} &= x_{6,t}^{[31..16]} \oplus x_{1,t}^{[15..0]} \end{aligned}$$

Here, $s_t^{[a..b]}$ denotes $(a-b+1)$-bit sub-block starting from the a-th bit (i.e. the most significant bit) to the b-th bit (i.e. the least significant bit), for $a \geq b$.

3 Bias of Rabbit

3.1 Related Work

Prior to our work, the bias of Rabbit was studied in [1] for a distinguishing attack. Recall that the bias of a binary random variable X is defined by $\epsilon(X) = \Pr(X = 0) - \Pr(X = 1)$. It was shown in [1] that $\epsilon(s_t^{[16k]}) > \epsilon(s_t^{[16k+j]})$ for $k = 0, \ldots, 7$ and $j = 1, \ldots, 15$. Furthermore, the bias for $s_t^{[0]}$, $s_t^{[32]}$, $s_t^{[64]}$ and $s_t^{[96]}$ was estimated to be $2^{-123.5}$. Since a bias ϵ leads to a distinguishing attack with minimum data complexity $O(1/\epsilon^2)$, this gave a distinguishing attack which requires about 2^{247} keystream sub-blocks.

To ease the computation, the above approximation result of [1] was based on the assumption that for the modular addition, the probability to return a carry bit at a given position is independent from the distribution in lower bit positions, which is not true. It was conjectured in [1] that this assumption would not weaken the results significantly. By comparison, as we will show immediately below, this assumption is *critical* for the computation, which cannot be ignored.

3.2 Our Results

We note that $x_{j,t}$ $(j = 0, \ldots, 8)$ is the modular addition of three independent g's. Thus, following [14], the distribution of $x_{j,t}$ can be computed efficiently by Fast Fourier Transform[1] (FFT) from the distributions of the three relevant g's.

More formally, we define 32-bit $y_{j,t} = x_{j,t} + c_{j,t+1} \mod 2^{32}$ for $j = 0, \ldots, 7$. We have

$$g_{j,t} = y_{j,t}^2 \oplus (y_{j,t}^2 \gg 32). \tag{11}$$

Assuming that $x_{j,t}$'s and $c_{j,t+1}$'s are uniformly and independently distributed, we know that so are $y_{j,t}$'s. This means the distributions of $g_{j,t}$'s for all j's are identical assuming that all the inputs are uniformly and independently distributed.

[1] See book [6] for introduction on FFT.

So we let $D(g)$ be the distribution of $g_{j,t}$ assuming the input $y_{j,t}$ is uniformly distributed for any j. Note that $D(g)$ can be easily computed by enumerating all possibilities in time $O(2^{32})$. From Eq.(1-8), it is clear that the distributions of $x_{j,t+1}$ with even j's are identical and we represent it by $D(x_0)$; similarly, the distributions of $x_{j,t+1}$ with odd j's are identical and we represent it by $D(x_1)$.

Following Eq.(1-8), $D(x_j)$ can be computed by FFT according to [14]. Let $D(g \lll 16)$ be the distribution of $g_{j,t} \lll 16$ (which is identical for all j's and thus omitted from our notation), and we have

$$D(x_0) = D(g) \otimes D(g \lll 16) \otimes D(g \lll 16),$$

where $\otimes$ denotes convolution and it is well known that it can be computed efficiently by FFT as follows

$$D(x_0) = \mathrm{FFT}^{-1}(\ \mathrm{FFT}(D(g)) \cdot \mathrm{FFT}(D(g \lll 16)) \cdot \mathrm{FFT}(D(g \lll 16))).$$

The multiplication of two arrays above denotes the element-wise multiplication. Similarly we can compute $D(x_1)$ by

$$D(x_1) = \mathrm{FFT}^{-1}(\ \mathrm{FFT}(D(g)) \cdot \mathrm{FFT}(D(g)) \cdot \mathrm{FFT}(D(g \lll 8))).$$

We computed $D(x_1)$. Our results show that $\epsilon(x_5^{[16]}) \approx 2^{-49.56}$ (Note that the precision is kept until the second last digit in the exponent). In comparison, it was estimated in [1] that $\epsilon(x_5^{[16]}) \approx 2^{-75.73}$, which is reduced by a dramatic factor of $2^{26.17}$ in our work. On the other hand, as pointed out in [1], $\epsilon(x_0^{[0]}) \approx 2^{-46.85}$. Assuming the independence of x_0 and x_5, we have

$$\epsilon(s_t^{[0]}) = \epsilon(x_0^{[0]}) \cdot \epsilon(x_5^{[16]}) \approx 2^{-96.41},$$

by Piling-up lemma [13]. This leads to a simple distinguishing attack with data complexity $1/\epsilon^2(s_t^{[0]})$, i.e. $O(2^{192.82})$, which is reduced by a factor of $2^{52.34}$ compared with [1].

Furthermore, for a sample distribution $\mathcal{D}$ with the sample size of $r \geq 1$ bits, it's shown in [2] that we can optimize the distinguisher by considering all the biases of $\mathcal{D}$ simultaneously with the minimum data complexity $O(1/\Delta(\mathcal{D}))$, where the Squared Euclidean Imbalance (SEI) is defined as $\Delta(\mathcal{D}) = 2^r \sum_a (\mathcal{D}(a) - 2^{-r})^2$. We have computed SEI for the relevant distributions. The results are summarized in Table 1 and Table 2 respectively.

Table 1. Our results on $D(g), D(x_1^{[31..16]}), D(x_1^{[15..0]}), D(x_1^{[31..0]}), D(x_0^{[15..0]}), D(s^{[15..0]})$

Distribution $\mathcal{D}$	$D(g)$	$D(x_1^{[31..16]})$	$D(x_1^{[15..0]})$	$D(x_1^{[31..0]})$	$D(x_0^{[15..0]})$	$D(s^{[15..0]})$
SEI $\Delta(\mathcal{D})$	1.01	2^{-76}	2^{-100}	2^{-45}	2^{-100}	2^{-158}

Table 2. Our results on $D(x_0^{[31..16]}), D(x_0^{[15..0]}), D(x_0^{[31..0]}), D(x_1^{[15..0]}), D(s^{[31..16]})$

Distribution $\mathcal{D}$	$D(x_0^{[31..16]})$	$D(x_0^{[15..0]})$	$D(x_0^{[31..0]})$	$D(x_1^{[15..0]})$	$D(s^{[31..16]})$
SEI $\Delta(\mathcal{D})$	2^{-76}	2^{-100}	2^{-45}	2^{-100}	2^{-160}

From Table 1, we know

$$\Delta(D(s^{[15..0]})) = \Delta(D(s^{[47..32]})) = \Delta(D(s^{[79..64]})) = \Delta(D(s^{[111..96]})) \approx 2^{-158}.$$

Similarly, from Table 2, we know

$$\Delta(D(s^{[31..16]})) = \Delta(D(s^{[63..48]})) = \Delta(D(s^{[95..80]})) = \Delta(D(s^{[127..112]})) \approx 2^{-160}.$$

Thus, we have a distinguishing attack with data complexity $O(2^{158})$ (resp. $O(2^{160})$), which considers samples of 16 bits from each keystream block s_t's. Compared with the previous case when only 1-bit samples of each keystream block is considered, we gain an improvement with a reduced factor 2^{35}. To summarize, our current best distinguishing attack so far has a complexity $O(2^{158})$, which is much closer to yet higher than the exhaustive search 2^{128}. In the next section, we will extend our distinguishing attack to a multi-frame key-recovery attack.

4 An Extended Multi-frame Attack on Rabbit

In this section, we assume that we know many frames of keystreams, and we also know the relations between their internal states x's. More formally, we consider m frames of keystreams $s^1, s^2, \ldots, s^m$ (where $s^i = s_1^i, s_2^i, \ldots, s_n^i$ for $i = 1, \ldots, m$ and s_t^i is one keystream block of 128 bits generated at time t and n is a parameter to be discussed later) generated by Rabbit. Let $x_{0,1}^i, \ldots, x_{7,1}^i$, $c_{0,1}^i, \ldots, c_{7,1}^i$ and $\phi_{7,1}^i$ be the initial state of Rabbit to generate s^i for $i = 1, \ldots, m$. We further assume that we know

$$x_{j,1}^1 \oplus x_{j,1}^i = d_{j,1}^i \tag{12}$$

$$x_{j,2}^1 \oplus x_{j,2}^i = d_{j,2}^i \tag{13}$$

for $j = 1, 3, 5, 7$ and $i = 2, \ldots, m$. We are interested in recovering the keys for all frames from the known $d_{j,1}^i, d_{j,2}^i$'s and keystreams s^i's as short as possible. Our attack consists of the following steps:

Step 1: solve $x_{j,1}^i$'s (resp. $x_{j,2}^i$'s) from $d_{j,1}^i$'s (resp. $d_{j,2}^i$'s).
Step 2: solve $g_{j,0}^i$'s (resp. $g_{j,1}^i$'s) from $x_{j,1}^i$'s (resp. $x_{j,2}^i$'s).
Step 3: recover $c_{j,2}^i$'s and $\phi_{7,2}^i$'s.
Step 4: recover the keys.

We will detail our attack step by step and discuss the attack complexities.

4.1 Step One: Recover $x^i_{j,1}, x^i_{j,2}$'s

Our aim is to recover $x^i_{j,1}$'s (resp. $x^i_{j,2}$s') from $d^i_{j,1}$'s (resp. $d^i_{j,2}$'s). As it's exactly the same to recover $x^i_{j,2}$'s, we will only focus on recovering $x^i_{j,1}$'s. Let us illustrate how to recover $x^i_{1,1}$'s from $d^i_{1,1}$'s using our analysis results in last section. According to Eq.(12), our problem becomes how to recover the L-bit $x^1_{1,1}$ ($L = 32$). The idea is to try exhaustively on $x^1_{1,1}$, deduce the other $x^i_{1,1}$'s (for $i = 2, \ldots, m$) from $d^i_{1,1}$'s. Then feed the m sequences of $x^i_{1,1}$'s to the distinguisher. Recall that we know the distribution of $x^i_{1,1}$ following the last section. From [2] and Table 1, we know that we need the minimum

$$m = \frac{4L \log 2}{2^{-45}} \approx 2^{51.5}$$

to distinguish m sequences generated by the correct $x^1_{1,1}$.

Due to symmetry of the problem, we can similarly recover $x^i_{j,1}$'s independently from the $d^i_{j,1}$'s for $j = 1, 3, 5, 7$. We solve the remaining $x^i_{j,1}$'s for $j = 0, 2, 4, 6$ from the keystreams s^i_1. The total time complexity of this step is $2 \times 4 \times 2^L \times 2^m$, i.e. $O(2^{86.5})$.

4.2 Step Two: Recover $g^i_{j,0}$'s, $g^i_{j,1}$'s

We focus on how to get $g^i_{j,1}$'s from $x^i_{j,2}$'s.

Guess-and-Determine Approach. An easy way to recover $g^i_{j,1}$'s from $x^i_{j,2}$'s is the following Guess-and-Determine method. For each frame i, we guess 64-bit $g^i_{1,1}, g^i_{7,1}$, and solve the remaining $g^i_{0,1}$, $g^i_{2,1}$, $g^i_{3,1}$, $g^i_{4,1}$, $g^i_{5,1}$, $g^i_{6,1}$ from $x^i_{1,2}$, $x^i_{2,2}$, $x^i_{3,2}$, $x^i_{4,2}$, $x^i_{5,2}$, $x^i_{6,2}$ respectively. We expect the 8 equations of $x^i_{j,2}$'s result in a unique solution on the $g^i_{j,1}$'s. Thus, this takes time $O(2^{64})$ for each frame. In total, we need time $m \times 2^{64}$, i.e. $O(2^{115.5})$.

Algebraic Approach. Now, we propose the following algebraic approach to solve Eq.(1-8) in order to determine $g_{j,1}$'s. Notice that those equations are highly symmetric. Adding Eq.(2,4,6,8) together, we have

$$\begin{aligned} &x_{1,2} + x_{3,2} + x_{5,2} + x_{7,2} = 2(g_{1,1} + g_{3,1} + g_{5,1} + g_{7,1}) + \\ &((g_{0,1} \lll 8) + (g_{2,1} \lll 8) + (g_{4,1} \lll 8) + (g_{6,1} \lll 8)) . \end{aligned} \tag{14}$$

Adding Eq.(1,3,5,7) together, we have

$$\begin{aligned} &x_{0,2} + x_{2,2} + x_{4,2} + x_{6,2} = (g_{0,1} + g_{2,1} + g_{4,1} + g_{6,1}) + \\ &((g_{0,1} \lll 16) + (g_{2,1} \lll 16) + (g_{4,1} \lll 16) + (g_{6,1} \lll 16)) + \\ &((g_{1,1} \lll 16) + (g_{3,1} \lll 16) + (g_{5,1} \lll 16) + (g_{7,1} \lll 16)) . \end{aligned} \tag{15}$$

In order to get $\beta_0 = g_{0,1} + g_{2,1} + g_{4,1} + g_{6,1}$ and $\beta_1 = g_{1,1} + g_{3,1} + g_{5,1} + g_{7,1}$ from Eq.(14,15), we first introduce the following property of Almost-Right-Distributivity of the bit-wise rotation over the modular addition.

Proposition 1 (Almost-Right-Distributivity). *Given L and any L-bit A, B, for any $1 \leq \ell \leq L$ we have*

$$(A \lll \ell) + (B \lll \ell) = ((A + B) \lll \ell) + \delta,$$

where the additions are additions modular 2^L and $\delta = u \times 2^\ell - v \mod 2^L$ and $u, v \in \{0, 1\}$.

The proof is trivial. As shown below, we note that this proposition allows to compute all the possible $(A \lll \ell) + (B \lll \ell)$ from $(A + B) \lll \ell$ or vice versa by guessing only a few choices of δ's. Now,from Proposition 1, we deduce the following

$$\begin{aligned} &(g_{0,1} \lll 8) + (g_{2,1} \lll 8) + (g_{4,1} \lll 8) + (g_{6,1} \lll 8) \\ &= ((g_{0,1} + g_{2,1} + g_{4,1} + g_{6,1}) \lll 8) + \delta_0, \end{aligned} \tag{16}$$

and $\delta_0 = u_0 \times 2^8 - v_0 \mod 2^{32}$ where $u_0, v_0 \in \{0, 1, 2\}$. Similarly, we have

$$\begin{aligned} &(g_{0,1} \lll 16) + (g_{2,1} \lll 16) + (g_{4,1} \lll 16) + (g_{6,1} \lll 16) \\ &= ((g_{0,1} + g_{2,1} + g_{4,1} + g_{6,1}) \lll 16) + \delta'_0, \end{aligned} \tag{17}$$

$$\begin{aligned} &(g_{1,1} \lll 16) + (g_{3,1} \lll 16) + (g_{5,1} \lll 16) + (g_{7,1} \lll 16) \\ &= ((g_{1,1} + g_{3,1} + g_{5,1} + g_{7,1}) \lll 16) + \delta_1, \end{aligned} \tag{18}$$

where $\delta'_0 = u'_0 \times 2^{16} - v'_0 \mod 2^{32}$, $\delta_1 = u_1 \times 2^{16} - v_1 \mod 2^{32}$ and $u'_0, u_1, v'_0, v_1 \in \{0, 1, 2\}$. Rewrite Eq.(14,15), we have

$$x_{1,2} + x_{3,2} + x_{5,2} + x_{7,2} = 2\beta_1 + (\beta_0 \lll 8) + \delta_0 \tag{19}$$

$$\begin{aligned} x_{0,2} + x_{2,2} + x_{4,2} + x_{6,2} = \beta_0 + (\beta_0 \lll 16) + (\beta_1 \lll 16) + \\ \delta'_0 + \delta_1 \end{aligned} \tag{20}$$

We have $3^2 \times 5^2 = 225$ possibilities[2] for the pair $(\delta_0, \delta'_0 + \delta_1)$. Since the left-hand sides of Eq.(19,20) are known, we can guess the pair $(\delta_0, \delta'_0 + \delta_1)$ and thus express Eq.(19,20) in terms of only β_0, β_1 now. We can try for all possible β_0, determine β_1 in Eq.(19) and check the validity of Eq.(20). We expect to obtain one unique β_0, β_1 for each possible $(\delta_0, \delta'_0 + \delta_1)$.

Adding Eq.(2,6) (resp. Eq.(4,8)) and subtracting β_1, we can deduce $(g_{0,1} \lll 8) + (g_{4,1} \lll 8)$ (resp. $(g_{2,1} \lll 8) + (g_{6,1} \lll 8)$). Now, we guess $g_{0,1}$ and determine $g_{4,1}$. Then, we get $\gamma_1 = g_{1,1} + g_{7,1}$ and $\gamma_2 = g_{3,1} + g_{5,1}$ by Eq.(2,6), and we get $\gamma_3 = (g_{6,1} \lll 16) + (g_{7,1} \lll 16)$ and $\gamma_4 = (g_{2,1} \lll 16) + (g_{3,1} \lll 16)$ from Eq.(1,5). Meanwhile, we know the following equality

$$\begin{aligned} &x_{0,2} + x_{2,2} - x_{4,2} - x_{6,2} \\ &= g_{0,1} - g_{4,1} + (g_{0,1} \lll 16) - (g_{4,1} \lll 16) + \\ &\quad ((g_{1,1} \lll 16) + (g_{7,1} \lll 16)) - ((g_{3,1} \lll 16) + (g_{5,1} \lll 16)) + \\ &\quad (g_{2,1} - g_{6,1}) - ((g_{2,1} \lll 16) - (g_{6,1} \lll 16)) \end{aligned} \tag{21}$$

[2] Note that it's not necessary to know δ'_0, δ_1 individually, which has $3^4 = 81$ possibilities. From Eq.(20) it suffices to know the sum $\delta'_0 + \delta_1$ for our purpose, which has $5^2 = 25$ possibilities instead.

holds. Again, we apply Proposition 1 to express

$$(g_{1,1} \lll 16) + (g_{7,1} \lll 16) = (\gamma_1 \lll 16) + \delta_2,$$
$$(g_{3,1} \lll 16) + (g_{5,1} \lll 16) = (\gamma_2 \lll 16) + \delta_3,$$

where $\delta_2 = u_2 \times 2^{16} - v_2 \mod 2^{32}$, $\delta_3 = u_3 \times 2^{16} - v_3 \mod 2^{32}$ and $u_2, u_3, v_2, v_3 \in \{0, 1\}$. Therefore, for each of the 2^4 possible pairs (δ_2, δ_3), we determine $(g_{2,1} - g_{6,1}) - ((g_{2,1} \lll 16) - (g_{6,1} \lll 16))$ in Eq.(21) as the remaining terms are all known. Similarly, from each of the 2^2 possible values $((g_{2,1} \lll 16) - (g_{6,1} \lll 16)) - ((g_{2,1} - g_{6,1}) \lll 16)$, we deduce the value of $\lambda - (\lambda \lll 16)$, where $\lambda = g_{2,1} - g_{6,1}$. With a precomputation table, which needs $O(2^{32})$ for precomputation time and memory, we get λ in time $O(1)$. From the known $(g_{2,1} \lll 8) + (g_{6,1} \lll 8)$, we apply Proposition 1 again to deduce $\lambda' = g_{2,1} + g_{6,1}$ by guessing 2^2 possible $(g_{2,1} \lll 8) + (g_{6,1} \lll 8) - (\lambda' \lll 8)$. We solve two linear equations to get $g_{2,1}, g_{6,1}$ from λ, λ'.

After we determine $g_{0,1}, g_{2,1}, g_{4,1}, g_{6,1}$, we solve $g_{7,1}$ (resp. $g_{3,1}$) from the known γ_3 (resp. γ_4); and we get $g_{1,1}$ (resp. $g_{5,1}$) from the known γ_1 (resp. γ_2).

We compute the total complexity of this step as follows. For one frame, it takes time $225 \times (2^{32} + 2^{32} \times 2^4 \times 2^2)$, i.e. $O(2^{46})$. Note that we gain a reduced factor of 2^{18} compared with the previous guess-and-determine method. For m frames, we need time $m \cdot 2^{46}$, i.e. $O(2^{97.5})$.

4.3 Step Three: Recover $c^i_{j,2}$'s, $\phi^i_{7,2}$'s from s^i_3's

Recall that we already know the $g^i_{j,1}$'s in our last step. In this step, we try to determine $y^i_{j,1} = x^i_{j,1} + c^i_{j,2} \mod 2^{32}$ for each i by Eq.(9). We computed $g^i_{j,1}$ in Eq.(11) for all possible 32-bit input $y^i_{j,1}$. We found out that for each possible output[3] $g^i_{j,1}$, in average there are 1.59 inputs $y^i_{j,1}$ mapping to the same output $g^i_{j,1}$. Since $x^i_{j,1}$'s are known, this means that we have a total of 1.59 possible $c^i_{j,2}$'s for each i and j. Thus, for a fixed i, we have $1.59^8 \approx 40$ possibilities for $g^i_{0,1}, \ldots, g^i_{7,1}$. Then, we guess for the remaining one-bit $\phi^i_{7,2}$. So, we only need to exhaustively try the $40 \times 2 = 80$ candidates to recover the 513-bit internal state at time $t = 2$ for each i. We expect to get a unique solution with one more keystream block s^i_3. Obviously, the time complexity of this step is $m \times 80$, i.e. $O(2^{58})$, which is dominated by m.

4.4 Step Four: Recover the Key

We already recover the full internal state at $t = 2$. We first guess 16-bit $\phi_{7,0}$, $\phi_{0,1}, \ldots, \phi_{7,1}$ and $\phi_{0,2}, \ldots, \phi_{6,2}$ and we compute $c_{j,0}$'s, $c_{j,1}$'s, $c_{j,2}$'s $(j = 0, \ldots, 7)$ and $\phi_{7,2}$ according to the update function of Rabbit's internal state. We check our guesses by verifying $c_{j,2}$'s and $\phi_{7,2}$. Thus we obtain the correct $c_{j,0}$'s, $c_{j,1}$'s.

[3] Note that the cardinality of the codomain of g is $0xA1681D78 \approx 2^{31.33}$ instead of $0x100000000$, i.e. about 36.95% of all possible 32 bits have no preimage.

Secondly, we recover the unique $x_{j,0}$'s from known $c_{j,1}$'s and $g_{j,0}$'s with the same method in Section 4.3 in negligible time $O(40)$. Now that we have both $x_{j,0}$ and $c_{j,0}$, we can compute the original $c_{j,0}$'s before mixing with $x_{j,0}$'s by $c_{j,0} = c_{j,0} \oplus x_{(j+4 \mod 8),0}$. Finally, by guessing the related 32-bit ϕ's, we can compute $c_{j,-1}, \ldots, c_{j,-4}$ backwards from $c_{j,0}$'s. According to Eq.(10) in Section 2.1, the 256-bit $c_{j,-4}$'s are highly redundant, which are obtained from 128-bit key K only. We expect to verify our guesses and obtain the correct key. Clearly, the complexity of this step is $m(2^{16} + 40 + 2^{32}) = 2^{32}m$, i.e. $O(2^{83.5})$.

4.5 Overall Complexity

We give in Table 3 the complexity of each step in our attack in this section. From Table 3, it's clear that the overall complexity is dominated by Step Two. To summarize, we needs precomputation and memory $O(2^{32})$ and run-time $O(2^{97.5})$ to recover the keys for $2^{51.5}$ frames, given the first three keystream blocks of each frame and $2^{51.5}$ $d^i_{j,1}$'s, $d^i_{j,2}$'s for $j = 1, 3, 5, 7$.

Table 3. Detailed complexities of our extended attack

step	precomputation	memory	time
1	-	-	$2^{86.5}$
2	2^{32}	2^{32}	$2^{97.5}$
3	-	-	2^{58}
4	-	-	$2^{83.5}$
Total	2^{32}	2^{32}	$2^{97.5}$

5 Conclusion

In this paper, based on the work [1], we continued the analysis of the keystream bias produced by Rabbit. We first computed the exact bias of the keystream sub-blocks by FFT. Our result leads to the best distinguishing attack with the complexity 2^{158}, compared with the complexity 2^{247} in [1]. This is much closer to yet still higher than the exhaustive search complexity 2^{128}, considering the key size of 128 bits. Meanwhile, our result indicates that the approximation assumption used in [1] for estimation of the bias is *critical* and cannot be ignored. Secondly, we extend our distinguishing attack to a multi-frame key-recovery attack, assuming that we know the relation between part of the internal states of all frames. Our attack uses $2^{51.5}$ frames and the first three keystream blocks of each frame. It takes memory $O(2^{32})$, precomputation $O(2^{32})$ and time $O(2^{97.5})$ to recover the keys for all frames. It is the first known key-recovery attack on Rabbit, though our attack assumption of knowing $2^{51.5}$ $d^i_{j,1}, d^i_{j,2}$ is unusually strong. As an independent result, we introduced an interesting property of Almost-Right-Distributivity of the bit-wise rotation over the modular addition for our algebraic analysis. This allows to solve the nonlinear yet symmetric equation system more efficiently for our problem.

Finally, it remains an open challenge to further extend our key-recovery attack to the general setting where only keystream outputs are available (i.e. without the knowledge of d's). We believe it would be possible to try the approach of differential fault analysis, or the differential cryptanalysis in stream ciphers in the recent work of [3] to get those d's.

Acknowledgment

This work was supported in part by the Ministry of Education of Singapore under grant T206B2204.

References

1. Aumasson, J.-P.: On a bias of Rabbit (January 2007), http://eprint.iacr.org/2007/033
2. Baignères, T., Junod, P., Vaudenay, S.: How far can we go beyond linear cryptanalysis? In: Lee, P.J. (ed.) ASIACRYPT 2004. LNCS, vol. 3329, pp. 432–450. Springer, Heidelberg (2004)
3. Biham, E., Dunkelman, O.: Differential cryptanalysis in stream ciphers(2007), http://eprint.iacr.org/2007/218
4. Boesgaard, M., Vesterager, M., Pedersen, T., Christiansen, J., Scavenius, O.: Rabbit: A new high-performance stream cipher. In: Johansson, T. (ed.) FSE 2003. LNCS, vol. 2887, pp. 307–329. Springer, Heidelberg (2003)
5. Boesgaard, M., Vesterager, M., Christensen, T., Zenner, E.: The stream cipher Rabbit, the ECRYPT stream cipher project - eSTREAM Report 2005/024 (2005), http://www.ecrypt.eu.org/stream/
6. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to algorithms. MIT Press, Cambridge (2001)
7. Cryptico A/S, Algebraic analysis of rabbit, 2003. White paper.
8. Cryptico A/S, Analysis of the key setup function in rabbit, White paper (2003)
9. Cryptico A/S, Hamming weights of the g-function, White paper (2003)
10. Cryptico A/S, Periodic properties of rabbit, White paper (2003)
11. Cryptico A/S, Second degree approximations of the g-function, White paper (2003)
12. Cryptico A/S, Security analysis of the IV-setup for rabbit, White paper (2003)
13. Matsui, M.: Linear cryptanalysis method for DES cipher, EUROCRYPT 1993. In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 386–397. Springer, Heidelberg (1994)
14. Maximov, A., Johansson, T.: Fast computation of large distributions and its cryptographic applications. In: Roy, B. (ed.) ASIACRYPT 2005. LNCS, vol. 3788, pp. 313–332. Springer, Heidelberg (2005)

Algebraic Attack on HFE Revisited

Jintai Ding[1], Dieter Schmidt[1], and Fabian Werner[2]

[1] University of Cincinnati
ding@math.uc.edu, dieter.schmidt@uc.edu
[2] Technical University of Darmstadt
fw@cccmz.de

Abstract. In this paper, we study how the algebraic attack on the HFE multivariate public key cryptosystem works if we build an HFE cryptosystem on a finite field whose characteristic is not two. Using some very basic algebraic geometry we argue that when the characteristic is not two the algebraic attack should not be polynomial in the range of the parameters which are used in practical applications. We further support our claims with extensive experiments using the Magma implementation of F_4, which is currently the best publicly available implementation of the Gröbner basis algorithm. We present a new variant of the HFE cryptosystems, where we project the public key of HFE to a space of one dimension lower. This protects the system from the Kipnis-Shamir attack and makes the decryption process avoid multiple candidates for the plaintext. We propose an example for a practical application on GF(11) and suggest a test challenge on GF(7).

Keywords: HFE, Gröbner basis, multivariate public key cryptosystem.

1 Introduction

The family of multivariate public key cryptosystems [16,4] is considered as one of the main candidates that have the potential to resist the future quantum computer attacks. MPKC's security relies on the fact that the direct attack, which we call the algebraic attack, needs to solve a set of multivariate quadratic equations, which is in general $\mathcal{NP}$-hard [8].

A major research topic in this area is the family of HFE cryptosystems. The HFE encryption systems were presented by Jacques Patarin at Eurocrypt'96 [15]. The fundamental idea is very similar to that of Matsumoto and Imai [13]. One selects a polynomial in a large field and then transforms it into a polynomial system over a vector space of a much smaller field. The first attack on HFE was presented by Kipnis and Shamir [11]. They lifted the public key back into the large field and attacked the system via a so-called MinRank [3] method. This attack was further improved by Courtois [2] using different ideas to solve the associated MinRank problem. The theoretical conclusion of these attacks is that, if one fixes the key parameter D of HFE (or more precisely $log(D)$) then the secret key can be found not in exponential but in polynomial time as the number

T.-C. Wu et al. (Eds.): ISC 2008, LNCS 5222, pp. 215–227, 2008.

n of variables increase. However these attacks were not fully substantiated by computer experiments.

Later on a direct attack on HFE with the new Gröbner basis methods like F_4 or F_5 did not show an exponential but a polynomial behavior [7,9]. Additionally, Faugère broke one of the challenges set by Patarin. This was later confirmed by Allen Steel with his Magma implementation of F_4 [12], whose performance is even better than the one used by Faugère. The overall conclusion seems to be that the HFE family of cryptosystems is not secure.

However, if we look more carefully at all current algebraic attacks, we see that all of them only deal with the case, where the finite field is exactly GF(2). A key point of these attacks is that the so called field equations

$$x_i^2 - x_i = 0, \qquad i = 1, \ldots, n$$

are used in the attack of the systems. If these field equations are not utilized, or more precisely could not be utilized efficiently, then the complexity of the algebraic attacks could be totally different. We first use some basic tools of algebraic geometry, including the idea of the so called solution at infinity [14], to argue that indeed the algebraic attacks should not work if the field equations are not fully utilized. We then support our claim by doing extensive experiments using the F_4 implementation in Magma, which is the best implementation that is publicly available.

The paper is arranged as follows. First we will briefly describe the HFE cryptosystem and the algebraic attacks. We then present a theoretical argument why the algebraic attack complexity will change if we do not utilize the field equations. In the next section we will show via computer experiments using the Magma implementation of the new Gröbner basis F_4 that the timing of the algebraic attack on simple cases of HFE should not be polynomial but should be exponential if we work on a field whose characteristic is not two. We will then present our challenge and give our conclusions.

2 The HFE Scheme

The HFE encryption scheme utilizes two finite fields. We denote the small field with q elements as $\mathbf{F}$, and $\mathbf{K}$ as the extension field of degree n over $\mathbf{F}$. Patarin recommended that the choice for HFE should be $q = 2$ and $n = 128$. Given a basis of $\mathbf{K}$ over $\mathbf{F}$, we can identify $\mathbf{K}$ with an n-dimensional vector space over $\mathbf{F}$ by $\varphi : \mathbf{K} \to \mathbf{F}^n$ and its inverse φ^{-1}. The design of HFE is based on a univariate polynomial $P(X)$ over $\mathbf{K}$ of the form

$$P(X) = \sum_{i=0}^{r-1} \sum_{j=i}^{r-1} p_{ij} X^{q^i + q^j} + \sum_{i=0}^{r_1} p_i X^{q^i} + p, \tag{1}$$

where the coefficients p_{ij}, p_i, p are randomly chosen from $\mathbf{K}$ and r, r_1 are small such that the degree of $P(X)$ is less than some fixed parameter D. The limitation

on the degree D of $P(X)$ is required so that it is possible to find the roots of $P(X)$ efficiently during the decryption, for example by using Berlekamp's algorithm.

Let

$$\bar{P}(x_1, \ldots, x_n) = T \circ \varphi \circ P \circ \varphi^{-1} \circ S(x_1, \ldots, x_n) \quad (2)$$

$$= (\bar{P}_1(x_1, \ldots, x_n), \ldots, \bar{P}_n(x_1, \ldots, x_n)), \quad (3)$$

where T and S are two randomly chosen invertible affine transformations on $\mathbf{F}^n$. The private key of the HFE scheme is formed by $P(X)$, S and T. The public key $\bar{P}(x_1, \ldots, x_n)$ consists of

$$\{\bar{P}_1(x_1, \ldots, x_n), \ldots, \bar{P}_n(x_1, \ldots, x_n)\},$$

which are n quadratic polynomials in the n variables in $\mathbf{F}$.

3 The Algebraic Attack

Let us assume that someone uses the HFE cryptosystem for encryption of a message or plaintext $(x'_1, \ldots, x'_n)$. What he or she does is to compute

$$(y'_1, \ldots, y'_n) = \bar{P}(x'_1, \ldots, x'_n),$$

the ciphertext, and sends it to the owner of the public key.

In order to attack HFE or any multivariate public key cryptosystem, an attacker has already the public key $\bar{P}$ and he or she also has access to the ciphertext $(y'_1, \ldots, y'_n)$. This means that if the attacker can solve the equation

$$\bar{P}(x_1, \ldots, x_n) = (y'_1, \ldots, y'_n),$$

the solution will give the attacker the plaintext $(x'_1, \ldots, x'_n)$ and he or she breaks the cryptosystem. Solving the set of equations above directly is called the algebraic attack.

The Gröbner basis method [1] is the classical method of solving multivariate polynomial equations. However, it is very slow in general. Recently major improvements have been made by Faugère [5,6] with his F_4 and F_5 algorithms. We will not give the details of the algorithms and refer the reader to the references instead.

Let us assume that we need to solve the set of equations

$$f_1(x_1, \ldots, x_n) = \cdots = f_n(x_1, \ldots, x_n) = 0,$$

over any field. When the solutions of this set of equations has dimension 0, or more precisely, when the system has only finitely many solutions (including the solutions over the extension field of the field we work on), the Gröbner basis algorithm finds a set of polynomials of the form

$$\{g_1(x_1, \ldots, x_n), g_2(x_2, \ldots, x_n), g_3(x_3, \ldots, x_n), \ldots, g_n(x_n)\}$$

such that the set of polynomials g_i and the set of polynomials f_i generate exactly the same ideal in the polynomial ring. Then one can find the solution by solving first the equation

$$g_n(x_n) = 0,$$

to find the value of x_n. One can now plug the value of x_n into

$$g_{n-1}(x_{n-1}, x_n) = 0$$

to find the value of x_{n-1}, and so on until all x_i are found.

In order for this process to work correctly, the Gröbner basis must be computed with respect to a special ordering, mostly called lex-order. Henceforth we mean "Gröbner basis in lex order" when we speak of Gröbner basis, because we want it to have the elimination property for actually solving the system.

Faugère and Joux showed that in the process of finding the Gröbner basis the degree of the polynomials that the Gröbner basis algorithm will generate should not be higher than $log(D)$. This makes the algorithm complexity to be polynomial once one fixes D, since $log(D)$ is very small due to the considerations for decryption.

4 The Algebraic Attack Revisited

Now we would like to do a careful analysis what role the field equations play in the algebraic attacks of HFE. In the case of $q = 2$, the field equations, which are also quadratic, are easily used in the computations of the Gröbner basis. But if we work in a bigger field, say GF(11), then the field equations

$$x_i^{11} - x_i = 0, \qquad i = 1, \ldots, n$$

are of degree 11. The field equations can only be utilized in the computation of the Gröbner basis if the degree of a polynomial is at least 11. This means that even dealing with a relatively small number of variables, like 32, the number of monomials of a degree 11 polynomial is already $\frac{(32+11)!}{11!32!}$, which is roughly 2^{32}. With our current memory capacity, if n is more than 64, the Gröbner basis algorithm can not really use the field equations, even if we try to add them to the set of equations we want to solve.

Before we go on further, we would like to make the following remark to clear the concepts that often cause confusions. Given a polynomial $f(x_1, \ldots, x_n)$ over $\mathbf{F}$, we have two different ways to look at it: One way is to look at it as an element in the polynomial ring $\mathbf{F}[x_1, \ldots, x_n]$, or we can look at it as an element in the function ring

$$\mathbf{F}[x_1, \ldots, x_n] / < x_1^q - x_1, \ldots, x_n^q - x_n > .$$

In the second case we identify x_i^q with x_i.

Let $f_1(x_1, \ldots, x_n) = 0, \ldots, f_n(x_1, \ldots, x_n) = 0$ be a set of n multivariate polynomial equations in n variables over $\mathbf{F}$. If we only want the solutions in $\mathbf{F}$, we actually need to solve the set of equations

$$f_1(x_1, \ldots, x_n) = 0, \ \ldots, \ f_n(x_1, \ldots, x_n) = 0, \ x_1^q - x_1 = 0, \ \ldots, \ x_n^q - x_n = 0.$$

In this case, we need to find the Gröbner basis for the ideal generated by the set of polynomials

$$f_1(x_1, \ldots, x_n), \ldots, f_n(x_1, \ldots, x_n), x_1^q - x_1, \ldots, x_n^q - x_n$$

in the ring $\mathbf{F}[x_1, \ldots, x_n]$. So we generally work on the ring $\mathbf{F}[x_1, \ldots, x_n]$, and if we want to work in the function ring we include the field equations.

Let us consider the case in which we do not take the field equations into account. Our key observation is that for any system of multivariate polynomial equations, if there are d different values for each variable (including the values in the extension field, or its algebraic closure), we should not be able to solve this system directly via the Gröbner basis algorithm with a maximum degree of this variable lower than d.

Proposition 1. *Let $f_1(x_1, \ldots, x_n) = 0, \ \ldots, \ f_n(x_1, \ldots, x_n) = 0$ be a set of n multivariate polynomial equations in n variables over $\mathbf{F}$; for each x_i, $1 \leq i \leq n$, if x_i has d different solutions $\beta_1, \ldots, \beta_d$ (including the ones in the algebraic closure of $\mathbf{F}$), the maximum degree of the corresponding Gröbner basis – in particular $g_n(x_n)$ – must have a degree higher or equal to d.*

Proof. We can prove it easily by contradiction. Suppose we get exactly d values for x_n by the equations generated by the f_i. If the degree of $g_n(x_n)$ is d' with $d' < d$, then we will have only d' values for x_n. This is impossible.

Similarly we have

Proposition 2. *Let $f_1(x_1, \ldots, x_n) = 0, \ \ldots, \ f_n(x_1, \ldots, x_n) = 0, \ x_1^q - x_1 = 0, \ \ldots, \ x_n^q - x_n = 0$ be the set of $2n$ multivariate polynomial equations in n variables over $\mathbf{F}$; for each x_i, $1 \leq i \leq n$, if x_i has d different solutions $\beta_1, \ldots, \beta_d$ in $\mathbf{F}$, the maximum degree of the corresponding Gröbner basis – in particular $g_n(x_n)$ – must have a degree higher or equal to d.*

Proof. We can prove it as in the proposition above.

So if we include the field equations, then we are indeed looking for solutions in the original field. If we do not include the field equations, we are actually looking for the solutions in the algebraic closure of the original field.

From the analysis above, we can also see that the minimum degrees a Gröbner basis (in lex order) needs to deal with in these two cases are very different, one is determined by the number of solutions in the original field and another one is determined by the extension field or algebraic closure.

Now let us move back to our case, the HFE cryptosystems. First, we know that T has no impact on the number of solutions, and it is also clear that S also has no impact on the number of solutions, because it is just a change of basis. Therefore the number of solutions of the public equations is determined by the number of solutions of the equations in the form of

$$P(X) - P' = 0$$

over the big field **K**. Also because S is a random transformation, we have, in general, a high probability that for each variable all solutions will not have the same value.

In the case that we include the field equations, then we are looking for solutions of the following equations

$$P(X) - P' = 0,$$
$$X^{q^n} - X = 0.$$

From the argument of Faugère and Joux that the degree of the algebraic attack using the new Gröbner basis is less or equal to $log(D)$, we can actually make the following conjecture:

Conjecture. The number of solutions to the public equation in the case of $q = 2$ for HFE in the field **F** is less or equal to $log(D)$.

This easily follows from the argument above with the assumption of Faugère and Joux's claims. We also note here that in their argument about the complexity, they implicitly used the field equations, namely the equation:

$$X^{q^n} - X = 0.$$

We also have that

Theorem 1. *If we do not include the field equation, the overall Gröbner basis algorithm (including algorithms like FGLM for switching the term ordering) has to deal with polynomials whose degree is at least equal to the number of solutions of the equation*

$$P(X) - P' = 0$$

in the algebraic closure of **K**.

From the theory of the functions over a finite field, we know that given any polynomial, we have a high probability that it is irreducible and therefore has the number of solutions, which is the same as its degree. But our case clearly is different in the sense that we know already it has at least one solution in the field **K**. From the general theory we estimate that the number of solutions of the equation

$$P(X) - P' = 0$$

in the algebraic closure of **K** should not be less than half of D statistically speaking. We will confirm this from experiments in section 5.1.

This implies that the minimum degree that a Gröbner basis needs to handle is at least $D/2$, and if D is $11^2+11 = 132$, we simply can not calculate the Gröbner basis because we can not store a polynomials with 32 variables of degree 66.

This also implies that the field equations in the case of $q = 2$ play a critical role in determining the algebraic attack complexity on HFE. However, as the characteristic increases it becomes much more difficult to utilize the field equations. Therefore, from the theoretical arguments given above, we expect (or more precisely we speculate) that for an HFE cryptosystem over GF(11) and with degree

$D = 132$, the algebraic attack should not be polynomial but rather exponential in the parameters we consider practical, that is for the range $n \leq 128$. We do not have precise theoretical arguments to prove such a statement, but we will try to confirm this speculation with our computer experiments.

We also would like to note here that Faugère and Joux's argument, stating that the degree of the polynomials which the Gröbner basis algorithm will generate should not be higher than $log(D)$, relies very much on using the field equations of characteristic two. Their argument will definitely fail if it is not the case of characteristic two. This can be shown by a very complicated combinatorial argument. Giving a detailed analysis is beyond the scope of this paper and we will present it in a separate paper.

5 Computer Experiments

Our experiments are split up in two parts. The first one is on the number of solutions in the algebraic closure and the second one is on the amount of time and memory it takes F_4 to calculate a Gröbner basis for different HFE systems. All experiments have been done on a computer at the Technical University of Darmstadt, Germany. The computer is a SunFire-280R which has an UltraSparc 1.2 GHz processor with 5120 MB of memory installed. The operating system is SunOS 5.8 (also called Solaris 8).

5.1 Experiment on the Number of Solutions

In order to verify the claim that the number of solutions of $P(X) - P' = 0$ in terms of X is generically at least $D/2$ we ran an experiment: First, we set up an HFE system and its hidden field polynomial $P(X)$. We then encrypted a random plaintext X' by finding $P' = P(X')$. Afterwards the program calculated $P(X) - P'$ and factored this polynomial. We did 800 test cases, 400 using $n = 17$ and 400 using $n = 19$. Not a single factorization contained a factor with a multiplicity higher than one, which means that the number of solutions in all 800 tests was exactly D which is trivially bigger than $D/2$.

5.2 Experiment of Solving Equations by F_4

Currently it is commonly accepted that the new Gröbner basis algorithm F_4 [5] and F_5 [6] are the most powerful tools to solve polynomial equations. Because F_4 is the only one which is publicly available, we used the Magma implementation of F_4 in order to see what the complexity of the algebraic attacks are indeed like.

We first generated the public key equations and then used Magma to try to find the Gröbner basis of this system. The experiments, as expected, produced the full triangular Gröbner basis in lex order. Our program then found all solutions and verified that indeed they included the correct solution. All experiments were done without using the field equations as this slows things down (see Fig. 6).

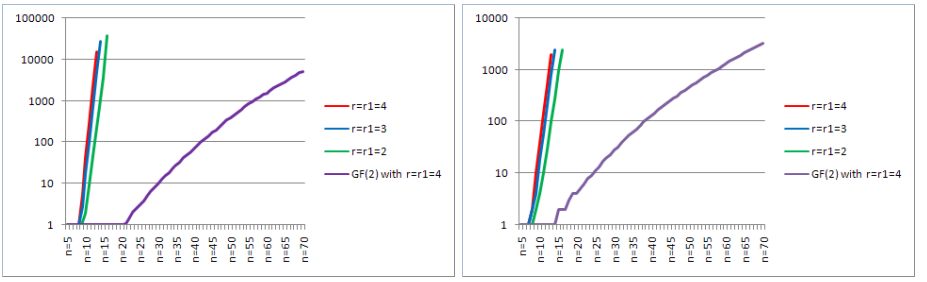


Fig. 1. Timings and memory usage for HFE systems over *GF(3)*

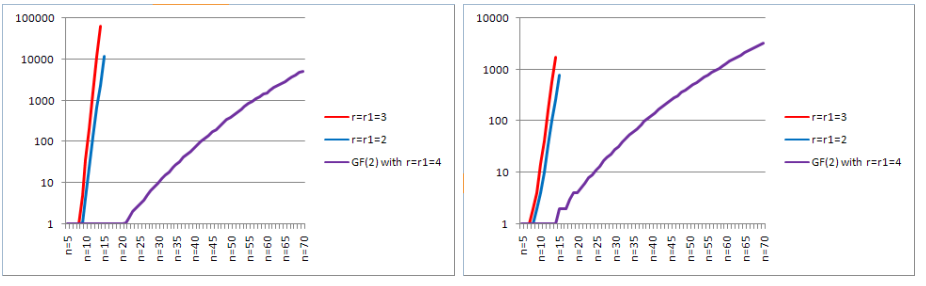


Fig. 2. Timings and memory usage for HFE systems over *GF(5)*

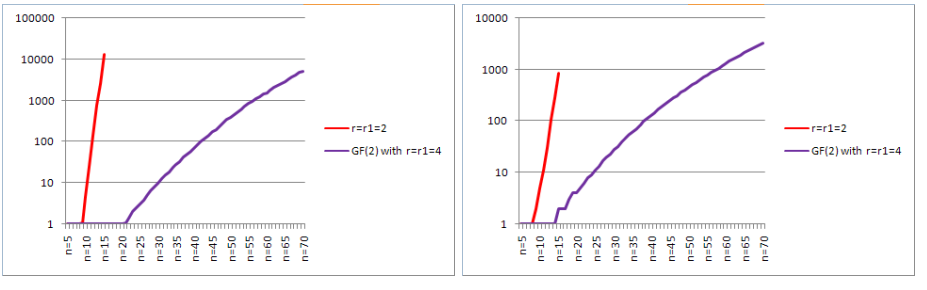


Fig. 3. Timings and memory usage for HFE systems over *GF(7)*

Tables below show the running time and the required memory of each n. In both figures we take n as the X-coordinate and show the running time (on the left, in seconds) and the required memory (on the right, in MB) as the logarithmic Y-coordinate. It clearly shows the exponential growing tendency with increasing n. The timing, we conclude, should be exponential and not polynomial. A more detailed overview over timings and memory usage can be found in the appendix. Much more theoretical and experimental work is still needed to fully understand the whole behavior.

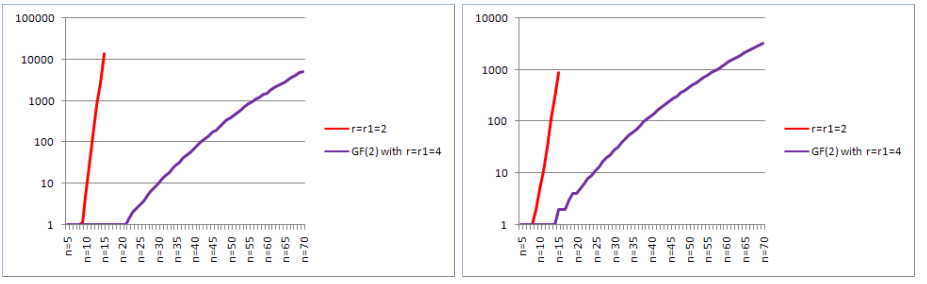


Fig. 4. Timings and memory usage for HFE systems over *GF(11)*

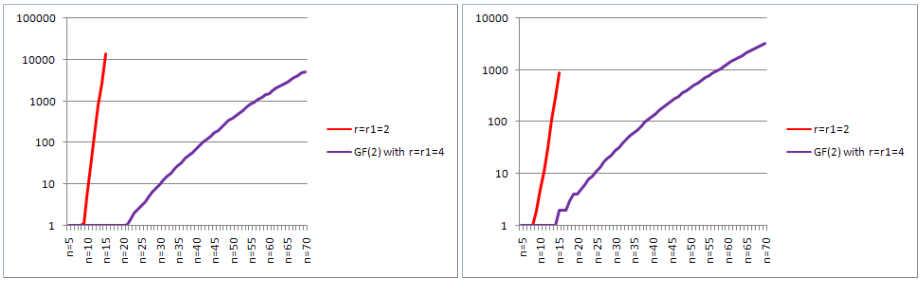


Fig. 5. Timings and memory usage for HFE systems over *GF(13)*

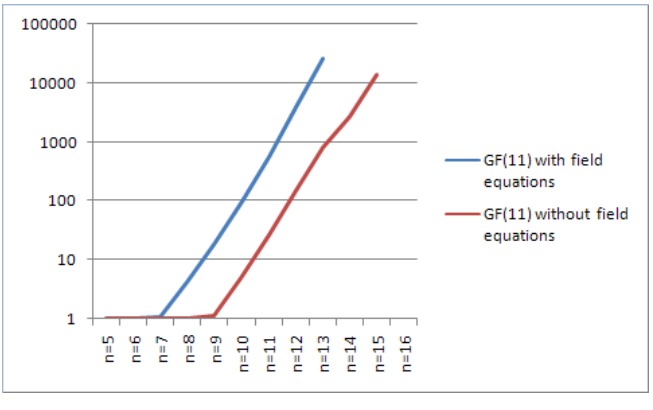


Fig. 6. Timings for HFE over *GF(11)* with and without field equations "$x_i^{11} - x_i$"

Currently it is not completely clear to us what the Magma F_4 implementation does when it comes to bigger characteristics. For GF(11), the implementation produced Gröbner bases, whose degree is higher than expected.

In order to see that the field equations do not help but even slow down the calculations, if they are not used properly, we re-ran the tests for GF(11) after putting in the field equations. The result also looks like expected, see Fig. 6.

6 New HFE Cryptosystems for Encryption

From the analysis above, we conclude that with a proper choice of parameters on the right field, we can build an HFE cryptosystem that could resist the algebraic attacks. But we also know that for any HFE cryptosystem, one must consider the Kipnis-Shamir attack. The recent work [10] actually shows that this attack does not work as efficiently as claimed. With this, we conclude one can build a reasonably secure HFE cryptosystem.

However, here we would like to propose a new type of HFE variant, which we call the projected HFE cryptosystem or PHFE.

Let $\bar{P}(x_1, \ldots, x_n)$ be the public key of HFE, then we randomly choose a linear equation

$$a_1 x_1 + \cdots + a_n x_n + a = 0. \tag{4}$$

We will pick a nonzero element among the a_i's, which we assume here to be a_n. Then we substitute x_n (or x_i) in $\bar{P}$ by the function

$$-\frac{a_1}{a_n}x_1 - \cdots - \frac{a_{n-1}}{a_n}x_{n-1} - \frac{a}{a_n},$$

which results in a new function:

$$\hat{P}(x_1, \ldots, x_{n-1}) = \bar{P}(x_1, \ldots, x_{n-1}, -\frac{a_1}{a_n}x_1 - \cdots - \frac{a_{n-1}}{a_n}x_{n-1} - \frac{a}{a_n}).$$

This will be the public key of PHFE. In this case, we have n polynomials and $n-1$ variables. The linear equation above also becomes part of the private key. The encryption process will be just as before. Decryption only varies in one point: once we have derived a few possible candidates for the plaintext, we will choose only the specific one, which satisfies the equation (4).

The new public key $\hat{P}$ can be seen as a projection of the old public key function $\bar{P}$. This projection map will serve two purposes:

1. It will destroy the hidden field structure of the old public key $\bar{P}$, such that the Kipnis-Shamir attack becomes useless, which is self-evident.
2. It will make the map more likely to be bijective so that the problem of multiple decryption choices becomes very unlikely.

This idea of projection was mentioned previously in several places, but it was never considered to be of any use because it does not help in terms of resisting the algebraic attacks.

Now we will take a look at the choices of a proper field $\mathbf{F}$. From our argumentation it seems, as if the system's security grows with the size of the ground field $\mathbf{F}$, but this does not work in all cases. By choosing $\mathbf{F}$ to have characteristic 2 and therefore cardinality 2^m, one can easily transform the public key into polynomials over GF(2). The only difference is an increment in the number of equations and the number of variables m. Then the algebraic attack still works as before when the degree of the polynomial $P(X)$ is not big enough. Therefore, we propose not to use a field of characteristic two.

For practical applications, we suggest that we should use GF(11) to build a PHFE system. We suggest D to be $11^2+11 = 132$ and $n = 89$, which should have the security level of at least 2^{80} triple DES from our estimation by computational experiments. In comparison with the HFE challenge broken by Faugère, in terms of memory, the public key of this new cryptosystem is about 5 times the size. In terms of the most costly part of the computation, namely the decryption process, the new system takes about twice the time to decrypt. All in all, the new system is comparable to the HFE challenge broken by Faugère.

To make the subject more interesting, we propose a test challenge, which, we speculate, might be within the reach of a practical attack with the most powerful computers of today. For the challenge we choose the field to be GF(7) with $D = 7^2 + 7 = 56$ and $n = 67$. The point is that if the claims about the algebraic attack on HFE with characteristic 2 is also valid here then one should be able to break our challenge.

7 Conclusion

We revisited the algebraic attack on the HFE cryptosystems. We showed that the algebraic attack on the HFE cryptosystems using the new Gröbner basis algorithm behaves differently, if it can not utilize the field equation to the full extent and the algebraic attack then can not work as efficiently as in the case of GF(2). Furthermore, we have shown via the new Gröbner basis algorithm F_4, that the complexity of the attack should be exponential and not polynomial when the characteristic of the field is not two. The key point of our theoretical argument is based on the simple idea that when solving a polynomial equation system, the degree parameter of the Gröbner basis algorithm is bounded from below by the number of solutions.

We also proposed a new variant of the HFE cryptosystems. The public key of HFE is projected to a space of one dimension lower. It serves the purpose to protect it form the Kipnis-Shamir attack and to avoid multiple candidates for the correct plaintext in the decryption process. We suggested an example for a practical application on GF(11), which we expect to be at the security level of 2^{80} triple DES, and a test challenge on GF(7) for practical attacks.

References

1. Buchberger, B.: Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal, Mathematical Institute, University of Innsbruck, Austria. Dissertation (1965)
2. Courtois, N.T.: The security of hidden field equations (HFE). In: Naccache, C. (ed.) CT-RSA 2001. LNCS, vol. 2020, pp. 266–281. Springer, Heidelberg (2001)
3. Courtois, N.T.: The Minrank Problem. MinRank, a new zero-knowledge scheme based on the NP-complete problem. Presented at the rump session of Crypto 2000, http://www.minrank.org
4. Ding, J., Gower, J., Schmidt, D.: Multivariate Public Key Cryptosystems. In: Advances in Information Security, Springer, Heidelberg (2006) (ISBN 0-387-32229-9)

5. Faugère, J.-C.: A New Efficient Algorithm for Computing Gröbner Bases (F_4). Journal of Pure and Applied Algebra 139, 61–88 (1999)
6. Faugère, J.-C.: A New Efficient Algorithm for Computing Gröbner Bases without Reduction to Zero(F_5). In: Mora, T. (ed.) Proceeding of ISSAC, pp. 75–83. ACM Press, New York (2002)
7. Faugère, J.-C., Joux, A.: Algebraic Cryptanalysis of Hidden Field Equation (HFE) Cryptosystems Using Gröbner Bases. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 44–60. Springer, Heidelberg (2003)
8. Garey, M.R., Johnson, D.S.: Computers and Intractability – A Guide to the Theory of NP-Completeness. W.H. Freeman and Company (1979) (ISBN 0-7167-1044-7 or 0-7167-1045-5)
9. Granboulan, L., Joux, A., Stern, J.: Inverting HFE Is Quasipolynomial. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 345–356. Springer, Heidelberg (2006)
10. Jiang, X., Ding, J., Hu, L.: Kipnis-Shamir's attack on HFE revisited Cryptology ePrint Archive (2007)
11. Kipnis, A., Shamir, A.: Cryptanalysis of the HFE public key cryptosystem by relinearization. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 19–30. Springer, Heidelberg (1999)
12. MAGMA Computational Algebra System, http://magma.maths.usyd.edu.au/magma/
13. Matsumoto, T., Imai, H.: Tsutomu Matsumoto and Hideki Imai. In: Günther, C.G. (ed.) EUROCRYPT 1988. LNCS, vol. 330, pp. 419–453. Springer, Heidelberg (1988)
14. Moh, T.-T.: On the method of "XL" and its inefficiency to TTM Cryptology ePrint Archive, Report 2001/047, http://eprint.iacr.org/
15. Patarin, J.: Hidden field equations (HFE) and isomorphism of polynomials (IP): Two new families of asymmetric algorithms. In: Maurer, U.M. (ed.) EUROCRYPT 1996. LNCS, vol. 1070. Springer, Heidelberg (1996)
16. Wolf, C., Preneel, B.: Taxonomy of public key schemes based on the problem of multivariate quadratic equations. Cryptology ePrint Archive, Report,2005/077, 12th of May 2005. 64 pages (2005), http://eprint.iacr.org/2005/077/

A Tables for GF(11) and GF(2)

TIMINGS	**GF(11)**	**GF(2)**
$n = 5$	0,01	0,01
$n = 6$	0,02	0,01
$n = 7$	0,07	0,01
$n = 8$	0,25	0,01
$n = 9$	1,13	0,01
$n = 10$	4,74	0,01
$n = 11$	25,87	0,02
$n = 12$	147,03	0,03
$n = 13$	799,96	0,04
$n = 14$	2722,40	0,13
$n = 15$	13744,27	0,17
$n = 16$	>84600	0,25
$n = 17$		0,32
$n = 18$		0,44
$n = 19$		0,61
$n = 20$		0,81
$n = 21$		1,05
$n = 22$		1,35
$n = 23$		1,94
$n = 24$		2,41
$n = 25$		3,03
$n = 30$		9,41
$n = 40$		70,98
$n = 50$		376,39
$n = 60$		1519,22
$n = 60$		1519,22
$n = 70$		4962,35

MEMORY USAGE	**GF(11)**	**GF(2)**
$n = 5$	0,76	0,67
$n = 6$	0,76	0,67
$n = 7$	0,95	0,67
$n = 8$	1,03	0,65
$n = 9$	1,06	0,72
$n = 10$	1,47	0,71
$n = 11$	2,88	0,74
$n = 12$	5,89	0,86
$n = 13$	14,23	0,93
$n = 14$	34,15	1,18
$n = 15$	105,76	1,35
$n = 16$		1,24
$n = 17$		2,55
$n = 18$		2,98
$n = 19$		1,72
$n = 20$		1,98
$n = 21$		2,21
$n = 22$		2,38
$n = 23$		2,52
$n = 24$		3,07
$n = 25$		3,45
$n = 30$		5,98
$n = 40$		16,20
$n = 50$		30,74
$n = 60$		59,57
$n = 70$		140,20

Revisiting Wiener's Attack – New Weak Keys in RSA

Subhamoy Maitra and Santanu Sarkar

Indian Statistical Institute, 203 B T Road, Kolkata 700 108, India
{subho,santanu_r}@isical.ac.in

Abstract. In this paper we revisit Wiener's method (IEEE-IT, 1990) of continued fraction (CF) to find new weaknesses in RSA. We consider RSA with $N = pq$, $q < p < 2q$, public encryption exponent e and private decryption exponent d. Our motivation is to find out when RSA is insecure given d is $O(n^\delta)$, where we are mostly interested in the range $0.3 \leq \delta \leq 0.5$. We use both the upper and lower bounds on $\phi(N)$ and then try to find out what are the cases when $\frac{t}{d}$ is a convergent in the CF expression of $\frac{e}{N-\frac{3}{\sqrt{2}}\sqrt{N}+1}$. First we show that the RSA keys are weak when $d = N^\delta$ and $\delta < \frac{3}{4} - \gamma - \tau$, where $2q - p = N^\gamma$ and τ is a small value based on certain parameters. This presents additional results over the work of de Weger (AAECC 2002). Further we show that, the RSA keys are weak when $d < \frac{1}{2}N^\delta$ and e is $O(N^{\frac{3}{2}-2\delta})$ for $\delta \leq \frac{1}{2}$. Using similar idea we also present new results over the work of Blömer and May (PKC 2004).

Keywords: Cryptanalysis, RSA, Factorization, Weak Keys.

1 Introduction

RSA [12] is one of the most popular cryptosystems in the history of cryptology. Here, we use the standard notations in RSA as follows: (i) primes p, q, with $q < p < 2q$; (ii) $N = pq$, $\phi(N) = (p-1)(q-1)$; (iii) $p - q = N^\beta$ where $N^{\frac{1}{4}} < N^\beta < \frac{N^{\frac{1}{2}}}{\sqrt{2}}$; (iv) e, d are such that $ed = 1 + t\phi(N)$, $t \geq 1$; (v) N, e are available in public and the message M is encrypted as $C = M^e \bmod N$; (vi) the secret key d is required to decrypt the message as $M = C^d \bmod N$.

In this paper we exploit the Wiener's method [18] of continued fraction (CF) to find new weaknesses in RSA (see [13] for Legendre's theorem related to CF expression). Wiener [18] showed that if $d < \frac{1}{3}N^{0.25}$, then $|\frac{e}{N} - \frac{t}{d}| < \frac{1}{2d^2}$ and $\frac{t}{d}$ (which in turn reveals p, q) could be estimated in $poly(\log N)$ time from the CF expression of the publicly available quantity $\frac{e}{N}$.

From $ed = 1 + t\phi(N)$, it is easy to see that $\frac{e}{\phi(N)} - \frac{t}{d} = \frac{1}{d\phi(N)}$, i.e., $\frac{e}{\phi(N)} - \frac{t}{d} < \frac{1}{2d^2}$ whenever $2d < \phi(N)$. Thus a good estimation of $\phi(N)$ can be of use while exploiting CF expression. It is known that for $q < p < 2q$, $N - \frac{3}{\sqrt{2}}\sqrt{N} + 1 < \phi(N) < N - 2\sqrt{N} + 1$. In [20, Section 4], Wiener's attack [18] has been extended

T.-C. Wu et al. (Eds.): ISC 2008, LNCS 5222, pp. 228–243, 2008.

estimating $\phi(N)$ as $N - 2\sqrt{N} + 1$. In our approach, both sides of the bound of $\phi(N)$ are exploited to get the results.

Lots of weaknesses of RSA have been identified in past three decades, but still RSA can be securely used with proper precautions as a public key cryptosystem. The security of RSA depends on the hardness of factorization. Let us now briefly discuss some weaknesses of RSA. RSA is found to be weak when the prime factors of either $p-1$ or $q-1$ are small [11]. Similarly, RSA is weak too when the prime factors of either $p+1$ or $q+1$ are small [19]. In [8], it has been pointed out that short public exponents may cause weakness if same message is broadcast to many parties. An outstanding survey on the attacks on RSA is available in [2]. For very recent results on RSA one may refer to [5,10,7] and the references therein.

In this paper we study the weaknesses of RSA when the secret decryption exponent d is upper bounded. The work of [18] initiates the application of Continued Fraction (CF) expression for the attack. In the work of [4], important results have been shown regarding small solutions to polynomial equations that in turn show vulnerabilities of low exponent RSA. In [3], the method of [4] has been exploited to show that RSA is insecure if $d < N^{0.292}$. The results from [4] have been used along with the results of [18] in many papers [3,20,1,9] to get the weaknesses when d is less than N^{δ}.

In this paper, we like to find out how the idea of CF expression from [18] can be exploited to find weaknesses of RSA when d is small. In [20, Section 4], some extension of the work [18] has been mentioned and it has also been noted that similar extension will work on the results of [17]. The result of [17] works for d with a few more bits longer than $N^{\frac{1}{4}}$. In [6], an extension of Legendre's result has been studied to get more weak keys in the direction of [17]. However, we find that new weak keys of RSA can be identified using the CF technique. These weak keys have not been explored in the literature before to the best of our knowledge.

In [18], it has been shown that RSA is not secure when $d < \frac{1}{3}N^{0.25}$ as under this condition, $|\frac{e}{N} - \frac{t}{d}| < \frac{1}{2d^2}$ and $\frac{t}{d}$ can be found in the CF expression of $\frac{e}{N}$. The knowledge of d helps in getting p, q immediately. In [16], a negative result has been identified that Wiener's attack will work with negligible success for $d > N^{\frac{1}{4}}$. Thus there is a deep interest to find out cases where the Wiener's strategy [18] can be extended to get more weak keys.

One may easily check that $\frac{e}{\phi(N)} > \frac{t}{d}$ and $\frac{e}{N} < \frac{t}{d}$. In [18], $\phi(N)$ has been approximated by N to get the results. A better result has been obtained in [20, Section 4] where $\phi(N)$ is approximated by $N - 2\sqrt{N} + 1$. It has been shown that $|\frac{e}{N-2\sqrt{N}+1} - \frac{t}{d}| < \frac{1}{2d^2}$ when $\delta < \frac{3}{4} - \beta$, where $p - q = N^{\beta}$ and $d = N^{\delta}$. Note that, for $\beta = \frac{1}{2}$, the result of [20] gives similar bound on d as in [18], which is of the order $N^{\frac{1}{4}}$. The improvement is obtained when β decreases. Only at $\beta = \frac{1}{4}$, d becomes of the order of $N^{\frac{1}{2}}$. In [20, Section 5, 6], the attack of [3] has been extended considering the value of β, where $p - q = N^{\beta}$. Instead of considering $p - q = N^{\beta}$, we here consider $2q - p = N^{\gamma}$ to get additional results. These results are presented in Section 2.

Further, instead of relating N^β, $\frac{1}{4} \leq \beta \leq \frac{1}{2}$, with $d = N^\delta$, we put the constraint on e. We find that RSA is insecure when d is of the order of N^δ for $\delta \leq 0.5$. The constraint in our case is on the public exponent e, which is related to the difference of the primes. We show that our attack works when $e \leq \frac{2N^{1-2\delta} - \frac{N}{N-A+1}}{\frac{N}{N-B} - \frac{N}{N-A+1}}$, which can be estimated as $O(N^{1.5-2\delta})$ in general. Here $A = \sqrt{N^{2\beta} + 4N}$ and $B = \frac{3}{\sqrt{2}}\sqrt{N}$. The conservative upper bound on e, i.e., $O(N^{1.5-2\delta})$, ignores the term $N^{2\beta}$ in A and thus the difference between the two primes does not come into the picture for the attack in general. These results are presented in Section 2.

In [1], it has been shown that p, q can be found in polynomial time for every N, e satisfying $ex + y = 0 \bmod \phi(N)$, with $x \leq \frac{1}{3}N^{\frac{1}{4}}$ and $|y| = O(N^{-\frac{3}{4}}ex)$; further some extensions considering the difference $p-q$ have also been considered. The work of [1] also uses the result of [4] as well as the idea of CF expression [18] in their proof. We also provide additional result over [1] using the lower bound of $\phi(N)$. This is presented in Section 3.

We here highlight the contribution of this paper with enumeration of the cases where we find new weak keys of RSA considering the CF expression of $\frac{e}{N - \frac{3}{\sqrt{2}}\sqrt{N}+1}$.

1. $d = N^\delta$ and $\delta < \frac{3}{4} - \gamma - \tau$, where $2q - p = N^\gamma$ and τ is a small value based on certain parameters.
2. $d < \frac{1}{2}N^\delta$ and e is $O(N^{\frac{3}{2}-2\delta})$ for $\delta \leq \frac{1}{2}$.
3. $ex+y = m\phi(N)$ for $m > 0$, $x \leq \frac{7}{4}N^{\frac{1}{4}}$, $|y| \leq cN^{-\frac{3}{4}}ex$, $c \leq 1$ and $p-q \geq cN^{\frac{1}{2}}$.
4. $ex + y = m\phi(N)$, for $m > 0$, $0 < x \leq \sqrt{\frac{3}{4l}}\sqrt{\frac{\phi(N)}{e}}\frac{N^{\frac{3}{4}}}{2q-p}$ for some positive integer l based on certain parameters and $|y| \leq \frac{2q-p}{\phi(N)N^{\frac{1}{4}}}ex$.

Before proceeding further, let us explain the Continued Fraction (CF) expression. We follow the material from [15, Chapter 5] for this. Given a positive rational number $\frac{a}{b}$, a finite CF expression of $\frac{a}{b}$ can be written as $q_1 + \frac{1}{q_2 + \frac{1}{q_3 + \ldots + \frac{1}{q_m}}}$ or in short $[q_1, q_2, q_3, \ldots q_m]$. As an example, take the rational number $\frac{34}{99}$. One can write this as follows in the CF expression: $\frac{34}{99} = 0 + \frac{1}{\frac{99}{34}} = 0 + \frac{1}{2 + \frac{31}{34}} = 0 + \frac{1}{2 + \frac{1}{\frac{34}{31}}} = 0 + \frac{1}{2 + \frac{1}{1 + \frac{3}{31}}} = 0 + \frac{1}{2 + \frac{1}{1 + \frac{1}{\frac{31}{3}}}} = 0 + \frac{1}{2 + \frac{1}{1 + \frac{1}{10 + \frac{1}{3}}}}$, and in short $[0, 2, 1, 10, 3]$. Consider a subsequence of $[0, 2, 1, 10, 3]$ as $[0, 2, 1]$. Note that $0 + \frac{1}{2 + \frac{1}{1}} = \frac{1}{3} = \frac{33}{99}$, which is very close to $\frac{34}{99}$, i.e., a subsequence of CF will give an approximation of the rational number. Given that a, b are t bit integers, the CF expression $[q_1, q_2, q_3, \ldots, q_m]$ of $\frac{a}{b}$ can be found in $O(poly(t))$ time and can be stored in $O(poly(t))$ space. Any initial subsequence of $[q_1, q_2, q_3, \ldots, q_m]$, i.e, $[q_1, q_2, q_3, \ldots, q_r]$, where $1 \leq r \leq m$ is called the convergent of $[q_1, q_2, q_3, \ldots, q_m]$. As example, $[0, 2, 1]$ is a convergent of $[0, 2, 1, 10, 3]$, i.e., $\frac{1}{3} = \frac{33}{99}$ is a convergent of $\frac{34}{99}$. Also note that if the subsequence has a 1 at the end then that may also written by adding the 1 to the previous integer and removing the 1. That is, both $[0, 2, 1]$ and $[0, 3]$ provides the same rational number.

2 New Weak Keys I

It is known that if $p - q < N^{\frac{1}{4}}$ [14] (see also [20, Section 3]), then RSA is weak by Fermat's factorization technique. Thus we are interested in the range $N^{\frac{1}{4}} < p - q < \frac{\sqrt{N}}{\sqrt{2}}$ only.

Proposition 1. *Let p, q be of same bit size, i.e., $q < p < 2q$. Then $\phi(N) > N - B + 1$, where $B = \frac{3}{\sqrt{2}}\sqrt{N}$. Further, if $p - q = N^{\beta}$ where $N^{\frac{1}{4}} < N^{\beta} < \frac{N^{\frac{1}{2}}}{\sqrt{2}}$, then $\phi(N) = N - A + 1$, where $A = \sqrt{N^{2\beta} + 4N}$.*

Proof. Since $(p - 2q)(2p - q) < 0$, we have $N - \frac{3}{\sqrt{2}}\sqrt{N} + 1 < \phi(N)$. Also, as $p - q = N^{\beta}$, we have $p^2 - N^{\beta}p - N = 0$, putting $q = \frac{N}{p}$. Thus $p = \frac{N^{\beta} + \sqrt{N^{2\beta}+4N}}{2}$. So we get $p + q = p + \frac{N}{p} = \frac{N^{\beta}+\sqrt{N^{2\beta}+4N}}{2} + \frac{2N}{N^{\beta}+\sqrt{N^{2\beta}+4N}} = \sqrt{N^{2\beta} + 4N}$. Then $\phi(N) = N - (p + q) + 1 = N - A + 1$. □

In [20], it has been identified that if $p - q = N^{\beta}$, then RSA is weak for $d = N^{\delta}$ when $\delta < \frac{3}{4} - \beta$. In such a case $\frac{t}{d}$ could be found as a convergent in the CF expression of $\frac{e}{N - 2\sqrt{N} + 1}$. Thus the result works better when p, q are close. As example, if $p - q = N^{\frac{1}{4}+\epsilon}$, then δ is bounded by $\frac{1}{2} - \epsilon$. As example, for $\epsilon = 0.05$, RSA becomes insecure if $d = N^{0.44} < N^{0.45}$. However, this improvement is not significant when $p - q$ is $O(N^{0.5})$. We present the following approach when $p - q$ is large, which gives $2q - p$ is small.

Proposition 2. *Let l be a positive integer. For $q > \frac{2l+2}{4l+1}p$, $|\frac{3}{\sqrt{2}}\sqrt{N} - (p + q)| < \frac{l(2q-p)^2}{(\frac{3}{\sqrt{2}}+2)\sqrt{N}}$.*

Proof. We have $\frac{1}{l}(\frac{3}{\sqrt{2}}\sqrt{N} - (p+q))(\frac{3}{\sqrt{2}}\sqrt{N} + (p+q)) < (2q-p)^2$ iff $((4l+1)q - (2l+2)p)(2q-p) > 0$. As $(2q-p) > 0$, we need $(4l+1)q - (2l+2)p > 0$. Thus, $q > \frac{2l+2}{4l+1}p$. Hence, $|\frac{3}{\sqrt{2}}\sqrt{N} - (p+q)| < \frac{l(2q-p)^2}{\frac{3}{\sqrt{2}}\sqrt{N}+(p+q)}$.

As $2\sqrt{N} < p + q < \frac{3}{\sqrt{2}}\sqrt{N}$, we have, $|\frac{3}{\sqrt{2}}\sqrt{N} - (p+q)| < \frac{l(2q-p)^2}{\frac{3}{\sqrt{2}}\sqrt{N}+2\sqrt{N}}$, which gives $|\frac{3}{\sqrt{2}}\sqrt{N} - (p+q)| < \frac{l(2q-p)^2}{(\frac{3}{\sqrt{2}}+2)\sqrt{N}}$. □

As example, for $l = 15$, we get $q > \frac{32}{61}p$. If l becomes larger then the constraint on q will almost reach the constraint that $q > \frac{1}{2}p$.

Theorem 1. *Let l be a positive integer, $q > \frac{2l+2}{4l+1}p$, $2q - p = N^{\gamma}$ and $d = N^{\delta}$. Then N can be factored in $O(poly(\log(N)))$ time when $\delta < \frac{3}{4} - \gamma - \tau$, where $2\tau > (\log \frac{4l}{\frac{3}{\sqrt{2}}+2})\frac{1}{\log N}$.*

Proof. Let $2q - p = N^{\gamma}$. Then $|\frac{3}{\sqrt{2}}\sqrt{N} - (p+q)| < \frac{lN^{2\gamma}}{(\frac{3}{\sqrt{2}}+2)\sqrt{N}}$,

i.e., $|\phi(N) - N - 1 + \frac{3}{\sqrt{2}}\sqrt{N}| < \frac{lN^{2\gamma}}{(\frac{3}{\sqrt{2}}+2)\sqrt{N}}$. Now,

$|\frac{e}{N-\frac{3}{\sqrt{2}}\sqrt{N}+1} - \frac{t}{d}| \leq |\frac{e}{N-\frac{3}{\sqrt{2}}\sqrt{N}+1} - \frac{e}{\phi(N)}| + |\frac{e}{\phi(N)} - \frac{t}{d}|$

$= \frac{e|\phi(N)-(N-\frac{3}{\sqrt{2}}\sqrt{N}+1)|}{\phi(N)(N-\frac{3}{\sqrt{2}}\sqrt{N}+1)} + \frac{1}{d\phi(N)} < \frac{e\frac{lN^{2\gamma}}{(\frac{3}{\sqrt{2}}+2)\sqrt{N}}}{\phi(N)(N-\frac{3}{\sqrt{2}}+1)} + \frac{1}{d\phi(N)}$

$< \frac{lN^{2\gamma}}{(\frac{3}{\sqrt{2}}+2)\sqrt{N}(N-\frac{3}{\sqrt{2}}+1)} + \frac{1}{d\phi(N)}$.

Assume, $N - \frac{3}{\sqrt{2}}\sqrt{N} + 1 > \frac{3}{4}N$ and $N > 8d$. Putting $d = n^{\delta}$, we get

$|\frac{e}{N-\frac{3}{\sqrt{2}}\sqrt{N}+1} - \frac{t}{d}| < \frac{lN^{2\gamma}}{(\frac{3}{\sqrt{2}}+2)\sqrt{N}\frac{3}{4}N} + \frac{4}{3Nd}$

$= \frac{\frac{4l}{3}N^{2\gamma-\frac{3}{2}}}{(\frac{3}{\sqrt{2}}+2)} + \frac{4}{3Nd} < \frac{\frac{4l}{3}N^{2\gamma-\frac{3}{2}}}{(\frac{3}{\sqrt{2}}+2)} + \frac{1}{6N^{2\delta}}$.

Note that $\frac{\frac{4l}{3}N^{2\gamma-\frac{3}{2}}}{(2+\frac{3}{\sqrt{2}})} < \frac{1}{3}N^{2\gamma-\frac{3}{2}+2\tau}$, for $2\tau > (\log \frac{4l}{\frac{3}{\sqrt{2}}+2})\frac{1}{\log N}$.

So, we get $|\frac{e}{N-\frac{3}{\sqrt{2}}\sqrt{N}+1} - \frac{t}{d}| < \frac{1}{3}N^{2\gamma-\frac{3}{2}+2\tau} + \frac{1}{6N^{2\delta}}$.

Thus, $|\frac{e}{N-\frac{3}{\sqrt{2}}\sqrt{N}+1} - \frac{t}{d}| < \frac{1}{2d^2}$, when $2\gamma - \frac{3}{2} + 2\tau < -2\delta$, i.e., $k < \frac{3}{4} - \gamma - \tau$.

In such a case, $\frac{t}{d}$ is a convergent of the CF expression of $\frac{e}{N-\frac{3}{\sqrt{2}+1}\sqrt{N}}$, and we can find it in $O(poly(\log(N)))$ time. □

In the above case, we consider the bound on $2q - p$ to extend the limit of d beyond $N^{0.3}$. The result of [20, Section 4] concentrated on the case when $p - q$ is bounded. On the other hand, our result does not consider the bound on $p - q$ and it works when $2q - p$ is bounded.

For practical implication, we work with primes $p, q \geq 10^{160}$, i.e., $N \geq 10^{320}$. It is clear that our attack will work better compared to existing works when p, q are away (i.e., p is close to $2q$) with the bound $q < p < 2q$. However, the experiments when $2q - p < n^{\frac{1}{4}}$ may not be of interest as in that case the factorization can be done in polynomial time similar to the argument of Fermat's factorization strategy [14] (see also [20, Section 3]). Thus we consider the scenario when $2q - p > N^{\frac{1}{3}}$. Below we present a practical example. All the examples in this paper involving large integers are implemented in LINUX environment using C with GMP.

Example 1. We choose a random prime $q \in [10^{160}, 10^{161}]$. Then we choose a random prime p, such that $2q-p > N^{\frac{1}{3}}$. In this example, $n^{0.346} < 2q-p < n^{0.347}$. We then choose the first d greater than or equal to N^{δ} for $\delta \geq \frac{1}{3}$ such that d is coprime to $\phi(N)$. In such a case if e is in our prescribed limit then our attack succeeds.

We consider p, q respectively as

213240012369375032891677978840508057002476631792587679131233694906832986110135 42

482710293984079429269505393966895473715804331857655334272013326966301014512312663 and

106620006184687516445838989420254028501238315896298839565616847453416493055067 71

241355146992039714634752696983447736857902165928827667136006663483150507256156183, which gives N as

227356514376456085145407643699497785267575964192664414706015616859113920770516 06

876372813657802669960516535143810533128200855625818799416971008924610927914638 14

72361264666736466411449942059568093916061632275622633234439324940363916123064654

02555303399548519028121978759763373757433442757741456334433042737747175925664532 9.

Note that $2q - p > N^{\frac{1}{3}}$. One can check that $\phi(N)$ is

22735651437645608514540764369949778526757596419266441470601561865911392077051606

87637281365780266996051653514381053312820085562581879941697100892461092791463814

40375262811330211477698245233491885365690137506733981364754270704338968206544340

30148759301936604637696169664729052700062792979093156193631043692802023748817648 4.

In Theorem 1, we require $q > \frac{2l+2}{4l+1}p$. Here this is satisfied for $l = 10^{50}$. Also we have $2\tau > (\log \frac{4l}{\frac{3}{\sqrt{2}}+2})\frac{1}{\log N}$, and it is enough to take $\tau = \frac{5}{64} = 0.078125$ in this case. Taking $\gamma = 0.347$ and $\tau = 0.078125$, we get $\delta < \frac{3}{4} - \gamma - \tau = 0.324875$. Thus, in this case for any $d < n^{0.324875}$, RSA will be insecure.

Now take $N^{0.32} < d < N^{0.324875}$. We consider $d =$

44138452180807132553854898960195837050529634687636859759755568727353610483058810149497334438480706535427

(a 104 digit number). The corresponding e is

85356738187677927267094758044990579754357485762742350715347494115752841684037367

61958050516985955514963349897936619515552408960795697318670660889152163280842447

75560973766638533120643123534024611720642739938697649334533161511773864127534483

56073872108358709307048969215446586611896268736369229047317637983628682308907311. The value of t is

16570953848141161450099797936855484729106684488828631895806571167212612482288825100679308747791603915419.

Here $\frac{t}{d}$ could be found in the CF expression of $\frac{e}{N - \lceil \frac{3}{\sqrt{2}}\sqrt{N} \rceil + 1}$ (see Appendix A). The | mark in the CF expression of $\frac{e}{N - \lceil \frac{3}{\sqrt{2}}\sqrt{N} \rceil + 1}$ points the termination of the subsequence for the CF expression of $\frac{t}{d}$).

In fact, Theorem 1 presents a sufficient condition on d when RSA will be weak. In Example 2, it is shown that even for some d, greater than the bound in Theorem 1, RSA can be insecure based on some condition on e. Example 2 shows that there exists some d even greater than $N^{\frac{1}{3}}$ when RSA is insecure. That is presented in the following discussion, where we try to remove the constraint on the difference between the primes; instead an upper bound on e is considered.

Lemma 1. *Let $2d < n^{\delta}$, where $0 < \delta \leq \frac{1}{2}$. Let A, B be as in Proposition 1. Then for $e \leq \frac{2N^{1-2\delta} - \frac{N}{N-A+1}}{\frac{N}{N-B} - \frac{N}{N-A+1}}$, it is possible to get $\frac{z_1}{z_2}$ such that*

1. $\frac{e}{N}\frac{z_1}{z_2} - \frac{t}{d} < \frac{1}{2d^2}$ *when* $\frac{N}{N-B+1} \leq \frac{z_1}{z_2} < \frac{2}{e}N^{1-2\delta} + \frac{N}{e}\frac{e-1}{N-A+1}$ *and*
2. $\frac{t}{d} - \frac{e}{N}\frac{z_1}{z_2} < \frac{1}{2d^2}$ *when* $\frac{N}{N-B+1} - \frac{2}{e}N^{1-2k} < \frac{z_1}{z_2} \leq \frac{N}{e}\frac{e-1}{N-A+1}$.

Proof. As we have $\frac{e}{N} < \frac{t}{d}$, there are two cases with the condition $\frac{z_1}{z_2} > 1$.

1. $\frac{t}{d} - \frac{e}{N} \geq \frac{1}{2d^2}$ but $0 \leq \frac{e}{N}\frac{z_1}{z_2} - \frac{t}{d} < \frac{1}{2d^2}$.
2. $\frac{t}{d} - \frac{e}{N} \geq \frac{1}{2d^2}$ but $0 \leq \frac{t}{d} - \frac{e}{N}\frac{z_1}{z_2} < \frac{1}{2d^2}$.

Case 1. The condition here is: $\frac{t}{d} - \frac{e}{N} \geq \frac{1}{2d^2}$ but $0 \leq \frac{e}{N}\frac{z_1}{z_2} - \frac{t}{d} < \frac{1}{2d^2}$.

Thus, we have to satisfy $0 \leq \frac{edz_1 - tNz_2}{Ndz_2} < \frac{1}{2d^2}$, i.e., $0 \leq \frac{z_1 + z_1 t\phi(N) - tNz_2}{Nz_2} < \frac{1}{2d}$.

Let $2d < N^\delta$, for $\delta > 0$. Then $0 \leq \frac{z_1+z_1t\phi(N)-tNz_2}{Nz_2} < \frac{1}{N^\delta}$ implies $0 \leq \frac{z_1+z_1t\phi(N)-tNz_2}{Nz_2} < \frac{1}{2d}$.

So we need to estimate $\frac{z_1}{z_2}$ considering $0 \leq \frac{z_1+z_1t\phi(N)-tNz_2}{Nz_2} < \frac{1}{N^\delta}$.

Now $0 \leq \frac{z_1+z_1t\phi(N)-tNz_2}{Nz_2} < \frac{1}{N^\delta}$ iff $0 \leq \frac{z_1}{z_2}(1+t\phi(N)) - tN < N^{1-\delta}$ iff $tN \leq \frac{z_1}{z_2}(1+t\phi(N)) < N^{1-k} + tN$ iff $\frac{tN}{1+t\phi(N)} \leq \frac{z_1}{z_2} < \frac{N^{1-\delta}+tN}{1+t\phi(N)}$ if $\frac{N}{\phi(N)} \leq \frac{z_1}{z_2} < \frac{N^{1-\delta}+tN}{ed}$ if $\frac{N}{N-B+1} \leq \frac{z_1}{z_2} < \frac{2}{e}N^{1-2\delta} + \frac{N}{e}\frac{e-1}{N-A+1}$, following (i) Proposition 1, (ii) $\frac{1}{d} > \frac{2}{N^\delta} \Rightarrow \frac{N^{1-\delta}}{ed} > \frac{2}{e}N^{1-2\delta}$, and (iii) $ed = 1 + t\phi(N) \Rightarrow \frac{t}{d} = \frac{e-\frac{1}{d}}{\phi(N)} \Rightarrow \frac{t}{d} > \frac{e-1}{N-A+1}$.

To have an $\frac{z_1}{z_2}$, we need $\frac{N}{N-B+1} < \frac{2}{e}N^{1-2\delta} + \frac{N}{e}\frac{e-1}{N-A+1}$.

For the guarantee of getting a rational $\frac{z_1}{z_2}$ in the interval $[\frac{N}{N-B+1}, \frac{2}{e}N^{1-2k} + \frac{N}{e}\frac{e-1}{N-A+1})$, one may choose $\frac{N}{N-\lceil B\rceil+1}$. Clearly, $\frac{N}{N-B+1} < \frac{N}{N-\lceil B\rceil+1} < \frac{N}{N-(B+1)+1} = \frac{N}{N-B}$. Thus, $\frac{N}{N-B} \leq \frac{2}{e}N^{1-2k} + \frac{N}{e}\frac{e-1}{N-A+1}$ need to be satisfied. This gives, $e \leq \frac{2N^{1-2\delta} - \frac{N}{N-A+1}}{\frac{N}{N-B} - \frac{N}{N-A+1}}$.

Case 2. The condition here is: $\frac{t}{d} - \frac{e}{N} \geq \frac{1}{2d^2}$ but $0 \leq \frac{t}{d} - \frac{e}{N}\frac{z_1}{z_2} < \frac{1}{2d^2}$. With similar analysis, we get $\frac{N}{N-B+1} - \frac{2}{e}N^{1-2\delta} < \frac{z_1}{z_2} \leq \frac{N}{e}\frac{e-1}{N-A+1}$, which again gives the same upper bound for e. □

Theorem 2. *Consider the interval I such that $I = (\frac{N}{N-B+1} - \frac{2}{e}N^{1-2\delta}, \frac{2}{e}N^{1-2\delta} + \frac{N}{e}\frac{e-1}{N-A+1})$. Let $2d < N^\delta$, where $0 < \delta \leq \frac{1}{2}$. Then for $e \leq \frac{2N^{1-2\delta} - \frac{N}{N-A+1}}{\frac{N}{N-B} - \frac{N}{N-A+1}}$, and $\frac{z_1}{z_2} \in I$, $|\frac{e}{N}\frac{z_1}{z_2} - \frac{t}{d}| < \frac{1}{2d^2}$.*

Proof. From Lemma 1 we get that $|\frac{e}{N}\frac{z_1}{z_2} - \frac{t}{d}| < \frac{1}{2d^2}$ for the intervals $\frac{N}{N-B+1} \leq \frac{z_1}{z_2} < \frac{2}{e}N^{1-2k} + \frac{N}{e}\frac{e-1}{N-A+1}$ and $\frac{N}{N-B+1} - \frac{2}{e}N^{1-2k} < \frac{z_1}{z_2} \leq \frac{N}{e}\frac{e-1}{N-A+1}$.

Since, $\frac{N}{N-B+1} - \frac{2}{e}N^{1-2\delta} < \frac{N}{e}\frac{e-1}{N-A+1} < \frac{N}{N-B+1} \leq \frac{z_1}{z_2} < \frac{2}{e}N^{1-2\delta} + \frac{N}{e}\frac{e-1}{N-A+1}$, it is enough to have $\frac{z_1}{z_2}$ in the interval $I = (\frac{N}{N-B+1} - \frac{2}{e}N^{1-2\delta}, \frac{2}{e}N^{1-2\delta} + \frac{N}{e}\frac{e-1}{N-A+1})$ to get $|\frac{e}{N}\frac{z_1}{z_2} - \frac{t}{d}| < \frac{1}{2d^2}$ for $2N^{1-\delta} \leq e < \frac{2N^{1-2\delta} - \frac{N}{N-A+1}}{\frac{N}{N-B} - \frac{N}{N-A+1}}$. □

Corollary 1. *Let $2d < N^\delta$, where $0 < \delta \leq \frac{1}{2}$ and $e \leq \frac{2N^{1-2\delta} - \frac{N}{N-A+1}}{\frac{N}{N-B} - \frac{N}{N-A+1}}$. Then N can be factored in poly$(\log N)$ time.*

Proof. The proof follows from Lemma 1 as $\frac{N}{N-B+1} < \frac{e}{N-\lceil B\rceil+1} < \frac{2}{e}N^{1-2\delta} + \frac{N}{e}\frac{e-1}{N-A+1}$. Then $\frac{t}{d}$ will be found in the CF expression of $\frac{e}{N}\frac{z_1}{z_2}$ when $\frac{z_1}{z_2} = \frac{N}{N-\lceil B\rceil+1}$. Thus $\frac{t}{d}$ will be found in the CF expression of $\frac{e}{N-\lceil B\rceil+1}$. □

Below we present the summarized result which is a conservative one as the upper bound of e is underestimated. This result is general as it does not require the parameter β for the proof, where $p - q = N^\beta$.

Theorem 3. *Let $N = pq$, where p, q are primes such that $q < p < 2q$. Then N can be factored in poly$(\log N)$ time from the knowledge of N, e when $d < \frac{1}{2}N^\delta$ and e is $O(N^{\frac{3}{2}-2\delta})$ for $\delta \leq \frac{1}{2}$.*

Proof. We have, $e \leq \frac{2N^{1-2\delta}-\frac{N}{N-A+1}}{\frac{N}{N-B}-\frac{N}{N-A+1}} = \frac{\left(2N^{-2\delta}(N-A+1)-1\right)(N-B)}{B-A+1}$, and this increases as A increases. Also the lower bound of A is $2\sqrt{N}$, when $N^{2\beta}$ is neglected. Thus, $e \leq \frac{2N^{1-2\delta}-\frac{N}{N-2\sqrt{N}+1}}{\frac{N}{N-\frac{3}{\sqrt{2}}\sqrt{N}}-\frac{N}{N-2\sqrt{N}+1}}$ and this is $O(N^{\frac{3}{2}-2\delta})$. □

The results given in Theorems 2, 3 do not put any constraint on the difference of the primes to get a better bound on d, but the constraint is imposed on e. When $d < \frac{1}{2}N^{\delta}$, then with increase in the value of δ, the value of e becomes upper bounded by $\frac{2N^{1-2\delta}-\frac{N}{N-A+1}}{\frac{N}{N-B}-\frac{N}{N-A+1}}$.

Theorem 3 shows that n can be factorized from the knowledge of e (d not known) when ed^2 is $O(n^{\frac{3}{2}})$ and d is $O(n^{\frac{1}{2}})$. We like to point out an important result [5, Theorem 2] that should be stated in this context, where it has been shown that for $ed \leq n^{\frac{3}{2}}$, with the knowledge of e, d, the integer n can be factorized in $O(\log^2 n)$ time.

In [20, Section 4], CF expression of only a specific value $\frac{e}{N-2\sqrt{N}+1}$ has been exploited to get $\frac{t}{d}$. Thus compared to our case, $\frac{z_1}{z_2}$ is approximated by $\frac{N}{N-2\sqrt{N}+1}$ in [20, Section 4]. Considering Lemma 1, if $\frac{N}{N-2\sqrt{N}+1} < \frac{N}{N-B+1} - \frac{2}{e}N^{1-2\delta}$, then the approach of [20] may not be used to get the primes, but our method will work.

The exact algorithm for our proposed attack is as follows.

Input: N, e, δ.
1. Compute the CF expression of $\frac{e}{N-\frac{3}{\sqrt{2}}\sqrt{N}+1}$.
2. For every convergent $\frac{t_1}{d_1}$ of the expression above if the roots of $x^2 - (N+1-\frac{ed_1-1}{t_1})x + N = 0$ are positive integers less than N then return the roots as p, q;
3. Return ("failure");

Our conservative estimate shows that the RSA keys are weak when $d < \frac{1}{2}N^{\delta}$ and e is $O(N^{\frac{3}{2}-2\delta})$. For example, considering $\delta = 0.3, 0.4, 0.45, 0.5$, e is bounded by $O(N^{0.9})$, $O(N^{0.7})$, $O(N^{0.6})$, $O(N^{0.5})$ respectively.

However, we like to point out that this is a conservative estimate and actually the upper bound of e is much better. We have $e \leq \frac{2N^{1-2\delta}-\frac{N}{N-A+1}}{\frac{N}{N-B}-\frac{N}{N-A+1}}$ and the attack works for $2d < n^{\delta}$. Thus the attack will work when $e \leq \frac{\frac{2N}{(2d+1)^2}-\frac{N}{N-A+1}}{\frac{N}{N-B}-\frac{N}{N-A+1}}$, taking $N^{\delta} = 2d+1$.

Example 2. Refer to p, q of Example 1. We consider $d > N^{\frac{1}{3}}$, which is

610336206651046900389953871563838676523222261232966853897231339740301854484426748686480182822423852911584 93

(a 107 digit number). The corresponding e is

256070337470608788319481009607488523602511607514442544529285221432548011674213622551315799000752368353532827651201521841634234079045126627056811374258890405913527886609642186978739480642254815290198948110261414415071190855304065173317461587219152177320300403509021656688133531875180594146046602509905386718318283402533.

Note that, we need to check $e \leq \frac{\frac{2N}{(2d+1)^2} - \frac{N}{N-A+1}}{\frac{N}{N-B} - \frac{N}{N-A+1}}$, taking $N^\delta = 2d+1$ and the value of $\frac{\frac{2N}{(2d+1)^2} - \frac{N}{N-A+1}}{\frac{N}{N-B} - \frac{N}{N-A+1}}$ is

277527825083860833403033559610727151722777672339402519575839704365461757777008185667568209319840663991545207478271466672207800668194684764406686250840094654048095827016310551668690003344650119766151234642917503628367993036711112155600249171858250543822132777886134760974691919179847616254071357103111675902815747786532,

which is greater than e indeed. The value of t is

687418167173701703542021027522201236378441184010988433098450217032667837807777970108983292838561347464 2.

Here $\frac{t}{d}$ could be found in the CF expression of $\frac{e}{N-\lceil \frac{3}{\sqrt{2}}\sqrt{N}\rceil+1}$ (see Appendix A). The | mark in the CF expression of $\frac{e}{N-\lceil \frac{3}{\sqrt{2}}\sqrt{N}\rceil+1}$ points the termination of the subsequence for the CF expression of $\frac{t}{d}$).

One may check that $\frac{t}{d}$ will not be found in the continued fraction expression of $\frac{e}{N}$ (Weiner's result [18]) or $\frac{e}{N-2\sqrt{N}+1}$ (Weger's result [20, Section 4]) in Example 2.

In [20, Sections 5, 6], the approach of [3] has been used to slightly improve the bounds of [20, Sections 4]. The improvement in that case is not evident when $p-q$ approaches $\sqrt{N}$ and it does not cover our results. In Example 2, $N^{0.4995} < p-q < N^{0.4996}$. Thus, for $p-q = N^\beta$, $\beta > 0.4995$. For $\beta = 0.4995$, we get $\delta < 1 - \sqrt{2\beta - \frac{1}{2}} = 0.2936$. Thus the method of [20, Section 6] will work for $d < N^{0.2936}$. Our example considers $d > N^{\frac{1}{3}}$ and hence not contained in the weak keys presented in [20, Section 6].

Remark 1. We also present Example 3 in Appendix A to show the effects of the upper bound on d in Theorem 1 as well as the upper bounds on d, e in Theorem 2. Note that

"d of Example 1" < "d of Example 3" < "d of Example 2".

For the "d of Example 3", $\frac{t}{d}$ cannot be found in the CF expression of $\frac{e}{N-\lceil \frac{3}{\sqrt{2}}\sqrt{N}\rceil+1}$. The "$d$ of Example 3" does not satisfy the condition given in Theorem 1. On the other hand, though "d of Example 3" < "d of Example 2", the bound on e is not satisfied in Example 3.

One may note that in Example 3, the CF expression of $\frac{t}{d}$ does not match only in only three places at the end with the initial subsequence of the CF expression of $\frac{e}{N-\lceil \frac{3}{\sqrt{2}}\sqrt{N}\rceil+1}$. Thus, the idea of search in the line of [17] will actually provide the exact result with some extra effort.

3 New Weak Keys II

Let us restate the result of [1, Theorem 2], where it was proved that p, q can be found in polynomial time for every N, e satisfying $ex + y = 0 \bmod \phi(N)$, with $x \leq \frac{1}{3}N^{\frac{1}{4}}$ and $|y| = O(N^{-\frac{3}{4}}ex)$.

Consider that $ex+y \equiv 0 \bmod \phi(N)$ and the interest is on the nontrivial cases. Thus $ex+y = m(N-p-q+1)$. This gives $\frac{e}{N} - \frac{m}{x} = -\frac{m(p+q-1)+y}{Nx}$. If $|\frac{e}{N} - \frac{m}{x}| = |\frac{m(p+q-1)+y}{Nx}| < \frac{1}{2x^2}$, then the fraction $\frac{m}{x}$ appears among the convergents of $\frac{e}{N}$. Thus one needs to find out the conditions such that $|m(p+q-1)+y| < \frac{N}{2x}$ is satisfied. Calculation shows that for $|y| = O(N^{-\frac{3}{4}}ex)$, one gets $x \leq \frac{1}{3}N^{\frac{1}{4}}$.

Note that instead of trying to find $\frac{m}{x}$ among the convergents of $\frac{e}{N}$, a better attempt will be to find $\frac{m}{x}$ among the convergents of $\frac{e}{\phi'(N)}$, where $\phi'(N)$ is a better estimate than N for $\phi(N)$. Following the idea of [20], $\phi'(N)$ has been taken as $N - \lfloor 2\sqrt{N} \rfloor$ (i.e., the upper bound of $\phi(n)$) and the CF expression of $\frac{e}{N-\lfloor 2\sqrt{N} \rfloor}$ has been considered to estimate $\frac{m}{x}$ in [1, Section 4]. It has been proved in [1, Theorem 4, Section 4] that p, q can be found in polynomial time for every N, e satisfying $ex+y = 0 \bmod \phi(N)$, with $x \leq \frac{1}{3}\sqrt{\frac{\phi(N)}{e}}\frac{N^{\frac{3}{4}}}{p-q}$ and $|y| \leq \frac{p-q}{\phi(N)N^{\frac{1}{4}}}ex$.

As we have done in the previous section, instead of considering the CF expression of $\frac{e}{N-\lfloor 2\sqrt{N} \rfloor}$, we consider the CF expression of $\frac{e}{N-\lceil \frac{3}{\sqrt{2}}\sqrt{N} \rceil+1}$ to get additional results.

Lemma 2. *Let $ex + y = m\phi(N)$ for $m > 0$. Then $|\frac{e}{N-\frac{3}{\sqrt{2}}\sqrt{N}+1} - \frac{m}{x}| < \frac{1}{2x^2}$ for $x \leq \frac{7}{4}N^{\frac{1}{4}}$ when $|y| \leq cN^{-\frac{3}{4}}ex$, where $c \leq 1$ and $p - q \geq cN^{\frac{1}{2}}$.*

Proof. Let us list the following observations.

1. From Proposition 1, we have $N - \frac{3}{\sqrt{2}}\sqrt{N} + 1 < \phi(N) < N - 2\sqrt{N} + 1$, which gives, $(2 - \frac{3}{\sqrt{2}})\sqrt{N} < p + q - \frac{3}{\sqrt{2}}\sqrt{N} < 0$. Thus, $|(2 - \frac{3}{\sqrt{2}})\sqrt{N}| > |p + q - \frac{3}{\sqrt{2}}\sqrt{N}|$, i.e., $(\frac{3}{\sqrt{2}} - 2)\sqrt{N} > |p + q - \frac{3}{\sqrt{2}}\sqrt{N}|$.
2. Also note that $|y| \leq cN^{-\frac{3}{4}}ex$, which gives $|y| < xN^{\frac{1}{4}}$ as $e < N$ and $c \leq 1$.
3. From [1, Proof of Theorem 2], $\frac{3}{4}\frac{ex}{\phi(N)} \leq m \leq \frac{5}{4}\frac{ex}{\phi(N)}$.

Now, $\frac{e}{N-\frac{3}{\sqrt{2}}\sqrt{N}+1} - \frac{m}{x} = \frac{m(-p-q+\frac{3}{\sqrt{2}}\sqrt{N})-y}{x(N-\frac{3}{\sqrt{2}}\sqrt{N}+1)}$.

This gives, $|\frac{e}{N-\frac{3}{\sqrt{2}}\sqrt{N}+1} - \frac{m}{x}| < \frac{m((\frac{3}{\sqrt{2}}-2)\sqrt{N})+|y|}{x(N-\frac{3}{\sqrt{2}}\sqrt{N}+1)}$ using item 1.

Now, $\frac{m((\frac{3}{\sqrt{2}}-2)\sqrt{N})+|y|}{x(N-\frac{3}{\sqrt{2}}\sqrt{N}+1)} < \frac{1}{2x^2}$ if $\frac{\frac{5}{4}\frac{ex}{\phi(N)}((\frac{3}{\sqrt{2}}-2)\sqrt{N})+xN^{\frac{1}{4}}}{x(N-\frac{3}{\sqrt{2}}\sqrt{N}+1)} < \frac{1}{2x^2}$ (using items 2, 3)

if $\frac{\frac{5}{4}x((\frac{3}{\sqrt{2}}-2)\sqrt{N})+xN^{\frac{1}{4}}}{x(N-\frac{3}{\sqrt{2}}\sqrt{N}+1)} < \frac{1}{2x^2}$ (as $\frac{e}{\phi(N)} < 1$) iff $\frac{\frac{5}{4}((\frac{3}{\sqrt{2}}-2)\sqrt{N})+N^{\frac{1}{4}}}{(N-\frac{3}{\sqrt{2}}\sqrt{N}+1)} < \frac{1}{2x^2}$

if $\frac{\frac{5}{4}\times 0.13\sqrt{N}}{(N-\frac{3}{\sqrt{2}}\sqrt{N}+1)} < \frac{1}{2x^2}$ (as $\frac{3}{\sqrt{2}} - 2 < 0.13$ and $\frac{5}{4}(\frac{3}{\sqrt{2}}-2)\sqrt{N})+N^{\frac{1}{4}} < \frac{5}{4}\times 0.13\sqrt{N}$ for large N)

iff $\frac{5}{2} \times 0.13x^2 < \sqrt{N} + \frac{1}{\sqrt{N}} - \frac{3}{\sqrt{2}}$
if $x^2 < 3.076\sqrt{N}$ (for large N) if $x \leq 1.75N^{\frac{1}{4}}$. □

This shows that the class of weak keys identified in [1, Theorem 2] can be extended by $\frac{21}{4}$, i.e., by more than 5 times.

In the improved result of [1, Theorem 4, Section 4], it has been shown that p, q can be found in polynomial time for every N, e satisfying $ex + y = 0 \bmod \phi(N)$, with $0 < x \leq \frac{1}{3}\sqrt{\frac{\phi(N)}{e}}\frac{N^{\frac{3}{4}}}{p-q}$ and $|y| \leq \frac{p-q}{\phi(N)n^{\frac{1}{4}}}ex$. Our result in Lemma 2 provides new weak keys which are not covered by the result of [1, Theorem 4, Section 4] in certain cases as follows.

Let $p - q = c\sqrt{N}$. As, $q < p < 2q$, we have $p - q < \sqrt{\frac{N}{2}}$. Thus, $c < \frac{1}{\sqrt{2}}$. In [1, Theorem 4, Section 4], it is given that $x \leq \frac{1}{3}\sqrt{\frac{\phi(N)}{e}}\frac{N^{\frac{3}{4}}}{p-q}$. Putting $p - q = c\sqrt{N}$, we find $x \leq \frac{1}{3c}\sqrt{\frac{\phi(N)}{e}}N^{\frac{1}{4}}$. Thus our result in Lemma 2 provides extra weak keys than [1, Theorem 4, Section 4] when $\frac{1}{3c}\sqrt{\frac{\phi(N)}{e}}N^{\frac{1}{4}} < \frac{7}{4}N^{\frac{1}{4}}$, which is true for $\frac{e}{\phi(N)} > \left(\frac{4}{21c}\right)^2$. As $e < \phi(N)$, $\frac{4}{21c} < 1$, which gives $c > \frac{4}{21}$. Thus the result our Lemma 2 presents new weak keys over In [1, Theorem 4, Section 4] when $\frac{e}{\phi(N)} > \left(\frac{4}{21c}\right)^2$ for $\frac{4}{21} < c < \frac{1}{\sqrt{2}}$.

Next we use our idea of considering $2q - p$ (as presented in Proposition 2) instead of $p - q$.

Theorem 4. *Let l be a positive integer such that $l > \frac{2(\frac{3}{\sqrt{2}}+2)}{\frac{3}{\sqrt{2}}-2\epsilon}$, where $\epsilon > \frac{2q-p}{\phi(N)N^{\frac{1}{4}}}$. Let $q > \frac{2l+2}{4l+1}p$. Suppose e satisfies the equation $ex + y = m\phi(N)$, for $m > 0$. Then N can be factored in $O(poly(\log(N)))$ time when $0 < x \leq \sqrt{\frac{3}{4l}}\sqrt{\frac{\phi(N)}{e}}\frac{N^{\frac{3}{4}}}{2q-p}$ and $|y| \leq \frac{2q-p}{\phi(N)N^{\frac{1}{4}}}ex$.*

Proof. We have $m = \frac{ex+y}{\phi(N)}$. Using the bound on $|y|$, we get $m \leq \frac{ex}{\phi(N)}\left(1 + \frac{2q-p}{\phi(N)N^{\frac{1}{4}}}\right)$. Now, $\left|\frac{e}{N-\frac{3}{\sqrt{2}}\sqrt{N}+1} - \frac{m}{x}\right| = \frac{|ex-m(N-\frac{3}{\sqrt{2}}\sqrt{N}+1)|}{x(N-\frac{3}{\sqrt{2}}\sqrt{N}+1)}$

$= \frac{|m(\frac{3}{\sqrt{2}}\sqrt{N}-p-q)-y|}{x(N-\frac{3}{\sqrt{2}}\sqrt{N}+1)}$ (putting $ex = -y + m\phi(N)$)

$\leq \frac{|m(\frac{3}{\sqrt{2}}\sqrt{N}-p-q)|+|y|}{x(N-\frac{3}{\sqrt{2}}\sqrt{N}+1)} < \frac{(\frac{ex}{\phi(N)}\left(1+\frac{2q-p}{\phi(N)N^{\frac{1}{4}}}\right))(\frac{l(2q-p)^2}{(\frac{3}{\sqrt{2}}+2)\sqrt{N}})+\frac{2q-p}{\phi(N)N^{\frac{1}{4}}}ex}{x(N-\frac{3}{\sqrt{2}}\sqrt{N}+1)}$ (putting the upper bound on m, using $\left|\frac{3}{\sqrt{2}}\sqrt{N} - (p+q)\right| < \frac{l(2q-p)^2}{(\frac{3}{\sqrt{2}}+2)\sqrt{N}}$ from Proposition 2 and the upper bound of y)

$= \frac{(\frac{e}{\phi(N)}\left(1+\frac{2q-p}{\phi(N)N^{\frac{1}{4}}}\right))(\frac{l(2q-p)^2}{(\frac{3}{\sqrt{2}}+2)\sqrt{N}})+\frac{2q-p}{\phi(N)N^{\frac{1}{4}}}e}{(N-\frac{3}{\sqrt{2}}\sqrt{N}+1)}$

$= \frac{\frac{e}{\phi(N)}\left((1+\frac{2q-p}{\phi(N)N^{\frac{1}{4}}})(\frac{l(2q-p)^2}{(\frac{3}{\sqrt{2}}+2)\sqrt{N}})+\frac{2q-p}{N^{\frac{1}{4}}}\right)}{(N-\frac{3}{\sqrt{2}}\sqrt{N}+1)}$

$\leq \frac{\frac{e}{\phi(N)}\left(\frac{l(2q-p)^2}{2\sqrt{N}}\right)}{(N-\frac{3}{\sqrt{2}}\sqrt{N}+1)}$, because of the following.

Let $X = \frac{2q-p}{N^{\frac{1}{4}}}$. Thus, $(1+\frac{2q-p}{\phi(N)N^{\frac{1}{4}}})(\frac{l(2q-p)^2}{(\frac{3}{\sqrt{2}}+2)\sqrt{N}}) + \frac{2q-p}{N^{\frac{1}{4}}} = (1+\frac{X}{\phi(N)})(\frac{lX^2}{\frac{3}{\sqrt{2}}+2} + X) < (1+\epsilon)(\frac{lX^2}{\frac{3}{\sqrt{2}}+2} + X) < \frac{l}{2}X^2$ if $l > \frac{2(\frac{3}{\sqrt{2}}+2)}{\frac{3}{\sqrt{2}}-2\epsilon}$, when $\epsilon > \frac{X}{\phi(N)}$, a very small quantity of $O(N^{-\frac{3}{4}})$. This is because, the numerator $2q-p$ is $O(N^{\frac{1}{2}})$ and the denominator contains $N^{\frac{1}{4}}\phi(N)$, where $\phi(N)$ is $O(N)$.

Assuming $N - \frac{3}{\sqrt{2}}\sqrt{N} > \frac{3}{4}N$, we get $|\frac{e}{N-\frac{3}{\sqrt{2}}\sqrt{N}+1} - \frac{m}{x}| < \frac{\frac{e}{\phi(N)}\left(\frac{l(2q-p)^2}{2\sqrt{N}}\right)}{\frac{3}{4}N}$.

Thus, $|\frac{e}{N-\frac{3}{\sqrt{2}}\sqrt{N}+1} - \frac{m}{x}| < \frac{1}{2x^2}$ if $\frac{\frac{e}{\phi(N)}\left(\frac{l(2q-p)^2}{2\sqrt{N}}\right)}{\frac{3}{4}N} \leq \frac{1}{2x^2}$

iff $0 < 2x^2 \leq \frac{\phi(N)}{e}\frac{\frac{3}{2}N^{\frac{3}{2}}}{l(2q-p)^2}$ iff $0 < x \leq \sqrt{\frac{3}{4l}}\sqrt{\frac{\phi(N)}{e}}\frac{N^{\frac{3}{4}}}{2q-p}$.

Given, $|\frac{e}{N-\frac{3}{\sqrt{2}}\sqrt{N}+1} - \frac{m}{x}| < \frac{1}{2x^2}$, N can be factorized using [1, Algorithm Generalized Wiener Attack II]. □

We have $l > \frac{2(\frac{3}{\sqrt{2}}+2)}{\frac{3}{\sqrt{2}}-2\epsilon}$. Now, $\frac{2(\frac{3}{\sqrt{2}}+2)}{\frac{3}{\sqrt{2}}} = 3.88561808316412673173$. Since 2ϵ is very small, one may assume $l = 4$ as a specific value. In such a case, $\sqrt{\frac{3}{4l}} > \frac{2}{5} > \frac{1}{3}$, when $q > \frac{10}{17}p$.

The result of [1, Theorem 4, Section 4] states that p, q can be found in polynomial time for every N, e satisfying $ex + y = 0 \bmod \phi(N)$, with $0 < x \leq \frac{1}{3}\sqrt{\frac{\phi(N)}{e}}\frac{N^{\frac{3}{4}}}{p-q}$ and $|y| \leq \frac{p-q}{\phi(N)n^{\frac{1}{4}}}ex$. In our result $p-q$ is replaced by $2q-p$. Thus the results of this section present new weak keys other than those presented in [1]. The result of [1, Theorem 4, Section 4] works efficiently when $p-q$ is upper bounded and our work gives better results when $2q-p$ is upper bounded.

4 Conclusion

In this paper we study the well known method of Continued Fraction (CF) expression to demonstrate new weak keys of RSA. The idea is to factorize n using the knowledge of e and some estimate of $\phi(N)$. One may note that in most of the cases $\frac{t}{d}$ can be found in the CF expression of $\frac{e}{\phi(N)}$. This idea was first proposed in [18], where the CF expression $\frac{e}{N}$ has been used to estimate $\frac{t}{d}$, i.e., N has been used as an estimate of $\phi(N)$. Later to that, $N - 2\sqrt{N} + 1$ (an upper bound of $\phi(N)$) has been used as an estimate of $\phi(N)$ in many works, e.g., [20,1]. In this paper we have studied both the upper and lower bounds of $\phi(N)$ carefully and used $N - \frac{3}{\sqrt{2}}\sqrt{N}+1$ (a lower bound of $\phi(N)$) as an estimate of $\phi(N)$. We extensively study the cases when $\frac{t}{d}$ can be found in the CF expression of $\frac{e}{N-\frac{3}{\sqrt{2}}\sqrt{N}+1}$. Our results provide new weak keys over the work of [20,1] and to the best of our knowledge the weak keys identified in our paper have not been presented earlier.

References

1. Blömer, J., May, A.: A generalized Wiener attack on RSA. In: Bao, F., Deng, R., Zhou, J. (eds.) PKC 2004. LNCS, vol. 2947, pp. 1–13. Springer, Heidelberg (2004)
2. Boneh, D.: Twenty Years of Attacks on the RSA Cryptosystem. Notices of the AMS 46(2), 203–213 (1999)
3. Boneh, D., Durfee, G.: Cryptanalysis of RSA with private key d less than $N^{0.292}$. IEEE Trans. on Information Theory 46(4), 1339–1349 (2000)
4. Copppersmith, D.: Small solutions to polynomial equations and low exponent vulnerabilities. Journal of Cryptology 10(4), 223–260 (1997)
5. Coron, J.-S., May, A.: Deterministic Polynomial-Time Equivalence of Computing the RSA Secret Key and Factoring. J. Cryptology 20(1), 39–50 (2007)
6. Duejella, A.: Continued fractions and RSA with small secret exponent. Tatra Mt. Math. Publ. 29, 101–112 (2004)
7. Jochemsz, E.: Cryptanalysis of RSA variants using small roots of polynomials. Ph. D. thesis, Technische Universiteit Eindhoven (2007)
8. Hastad, J.: On using RSA with low exponent in public key network. In: Williams, H.C. (ed.) CRYPTO 1985. LNCS, vol. 218, pp. 403–408. Springer, Heidelberg (1986)
9. Ibrahim, D., Bahig, H.M., Bhery, A., Daoud, S.S.: A new RSA vulnerability using continued fractions. In: 6th ACS/IEEE International Conference on Computer Systems and Applications (AICCSA 2008), Doha, Qatar, March 31–April 4 (2008)
10. Jochemsz, E., May, A.: A Polynomial Time Attack on RSA with Private CRT-Exponents Smaller Than $N^{0.073}$. In: Menezes, A. (ed.) CRYPTO 2007. LNCS, vol. 4622, pp. 395–411. Springer, Heidelberg (2007)
11. Pollard, J.M.: Theorems on factorization and primality testing. Proc. of Combridge Philos. Soc. 76, 521–528 (1974)
12. Rivest, R.L., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public key cryptosystems. Communications of ACM 21(2), 158–164 (1978)
13. Rosen, K.H.: Elementary Number Theory. Addison-Wesley, Reading (1984)
14. Silverman, R.D.: Fast generation of random, strong RSA primes. Cryptobytes 3(1), 9–13 (1997)
15. Stinson, D.R.: Cryptography – Theory and Practice, 2nd edn. Chapman & Hall/CRC, Boca Raton (2002)
16. Steinfeld, R., Contini, S., Pieprzyk, J., Wang, H.: Converse results to the Wiener attack on RSA. In: Vaudenay, S. (ed.) PKC 2005. LNCS, vol. 3386, pp. 184–198. Springer, Heidelberg (2005)
17. Verheul, E.R., van Tilborg, H.C.A.: Cryptanalysis of 'less short' RSA secret exponents. Applicable Algebra in Engineering, Communication and Computing 8, 425–435 (1997)
18. Wiener, M.: Cryptanalysis of short RSA secret exponents. IEEE Transactions on Information Theory 36(3), 553–558 (1990)
19. Williams, H.C.: A $p+1$ method of factoring. Mathematics of Computation 39(159), 225–234 (1982)
20. de Weger, B.: Cryptanalysis of RSA with small prime difference. Applicable Algebra in Engineering, Communication and Computing 13(1), 17–28 (2002)

Appendix A

Example 1. The CF expression of $\frac{e}{N-\lceil \frac{3}{\sqrt{2}}\sqrt{N}\rceil+1}$ is as follows.

0, 2, 1, 1, 1, 35, 1, 1, 1, 1, 4, 1, 1, 2, 11, 1, 3, 1, 1, 3, 2, 8, 30, 1, 1, 1, 16, 1, 1, 1, 1, 7, 1, 5, 1, 2, 1, 1, 1, 2,
1, 3, 1, 1, 1, 1, 2, 4, 2, 5, 1, 6, 1, 1, 1, 5, 4, 31, 7, 4, 1, 5, 5, 3, 1, 145, 1, 54, 5, 1, 4, 3, 2, 18, 1, 1, 1, 2, 1, 3,
3, 11, 6, 1, 1, 1, 1, 27, 4, 2, 1, 5, 1, 1, 3, 1, 11, 4, 3, 10, 1, 2, 1, 2, 3, 8, 1, 1, 1, 2, 1, 7, 1, 2, 3, 4, 1, 6, 3, 1,
4, 1, 8, 621, 1, 4, 2, 11, 1, 1, 35, 1, 113, 7, 1, 13, 1, 2, 1, 20, 1, 2, 6, 2, 1, 5, 3, 4, 1, 2, 17, 3, 2, 3, 3, 1, 1, 1, 2, 4,
1, 22, 1, 1, 4, 1, 1, 4, 1, 1, 3, 3, 1, 150, 4, 1, 1, 4, 2, 1, 1, 1, 9, 6, 1, 1, 1, 8, 1, 1, 30, 26, 1, 1, 1, 1, 9, 1, 6, 3,
3, 12, 1, 1, 1, 2, 2, 1, 14, 1, 3, 7, 1, 2, 1 |, 1242, 1, 1, 1, 2, 1, 4, 5, 12, 1, 1, 4, 13, 5, 4, 10, 1, 1, 1, 12, 1, 30, 2, 65, 10,
1, 2, 3, 1, 6, 1, 1, 15, 14, 6, 2, 9, 3, 2, 13, 2, 10, 1, 7, 1, 2, 1, 1, 1, 2, 1, 1, 1, 1, 1, 2, 17, 1,4, 1, 33, 1, 2, 5, 5,
26, 2, 1, 1, 3, 1, 1, 1, 1, 1, 1, 1, 4, 2, 1, 1, 48, 1, 1, 136, 1, 17, 1, 3, 1, 9, 6, 14, 1, 24, 2, 4, 31, 2, 2, 1, 1, 2, 2,
1, 3, 1, 2, 2, 1, 1, 1, 10, 1, 20, 7, 12, 3, 6, 1, 2, 5, 5, 1, 2, 5, 1, 1, 1, 3, 3, 1, 11, 8, 3, 2, 1, 75, 1, 1, 34, 1, 1, 3,
7, 2, 1, 2, 7, 1, 5, 1, 5, 1, 1, 16, 1, 1, 4, 2, 14, 1, 2, 8, 6, 6, 1, 1, 5, 1, 1, 2, 1, 1, 2, 523, 4, 1, 6, 1, 1, 2, 1, 4,
2, 1, 1, 1, 1, 2, 2, 11, 7, 1, 2, 28, 21, 1, 8, 11, 3, 1, 18, 1, 2, 1, 47, 1, 5, 1, 10, 2, 9, 1, 2, 3, 18, 1, 2, 1, 2, 7, 1, 6,
5, 3, 3, 14, 1, 1, 3, 2, 1, 1, 1, 2, 1, 10, 2, 2, 3, 3, 4, 1, 1, 2, 1, 4, 2, 1, 1, 3, 3, 1, 2, 1, 1, 1, 11, 1, 3, 1, 257, 2,
3, 5, 2, 1, 10, 1, 2, 2, 1, 1, 7, 1, 1, 2, 1, 4, 1, 10, 8, 3, 5, 1, 3, 1, 5, 1, 1, 1, 2, 4, 4, 2, 45, 1, 2, 60, 3, 1, 1, 1,
1, 5, 4, 3, 1, 2, 1, 1, 1, 15, 4, 2, 1, 1, 1, 1, 1, 1, 20, 3, 1, 4, 1, 1, 7, 3, 1, 4, 1, 1, 2, 5, 1, 3, 1, 2, 1, 1, 1, 1,
1, 28, 3, 49, 9, 9, 13, 7, 4, 3, 5, 2, 17, 1, 8, 1, 2, 2, 4, 5, 1, 1, 5, 1, 94, 1, 6, 1, 3, 1, 2, 1, 1, 12, 6, 1, 2, 1, 114, 2,
2, 24, 2, 3, 155, 1, 7, 1, 2, 1, 2, 19, 1, 9, 1, 6, 1, 3, 1, 1, 1, 1, 2, 2, 6, 1, 4, 1, 1, 5, 1, 2, 6, 1, 4, 1, 8, 1, 1, 1,
2, 84, 3.

The CF expression of $\frac{t}{d}$ is as follows.

0, 2, 1, 1, 1, 35, 1, 1, 1, 1, 4, 1, 1, 2, 11, 1, 3, 1, 1, 3, 2, 8, 30, 1, 1, 1, 16, 1, 1, 1, 1, 7, 1, 5, 1, 2, 1, 1, 1, 2,
1, 3, 1, 1, 1, 1, 2, 4, 2, 5, 1, 6, 1, 1, 1, 5, 4, 31, 7, 4, 1, 5, 5, 3, 1, 145, 1, 54, 5, 1, 4, 3, 2, 18, 1, 1, 1, 2, 1, 3,
3, 11, 6, 1, 1, 1, 1, 27, 4, 2, 1, 5, 1, 1, 3, 1, 11, 4, 3, 10, 1, 2, 1, 2, 3, 8, 1, 1, 1, 2, 1, 7, 1, 2, 3, 4, 1, 6, 3, 1,
4, 1, 8, 621, 1, 4, 2, 11, 1, 1, 35, 1, 113, 7, 1, 13, 1, 2, 1, 20, 1, 2, 6, 2, 1, 5, 3, 4, 1, 2, 17, 3, 2, 3, 3, 1, 1, 1, 2, 4,
1, 22, 1, 1, 4, 1, 1, 4, 1, 1, 3, 3, 1, 150, 4, 1, 1, 4, 2, 1, 1, 1, 9, 6, 1, 1, 1, 8, 1, 1, 30, 26, 1, 1, 1, 1, 9, 1, 6, 3,
3, 12, 1, 1, 1, 2, 2, 1, 14, 1, 3, 7, 1, 3.

Example 2. The CF expression of $\frac{e}{N-\lceil \frac{3}{\sqrt{2}}\sqrt{N}\rceil+1}$ is as follows.

0, 8878, 1, 2, 14, 12, 1, 1, 1, 3, 18, 1, 54, 2, 7, 10, 1, 2, 4124, 1, 1, 1, 168, 22, 9, 3, 1, 1, 8, 1, 2, 1, 1, 4, 2, 2, 1, 1, 4, 3,
1, 1, 1, 9, 2, 1, 1, 1, 206, 1, 11, 1, 9, 4, 39, 3, 1, 86, 1, 2, 1, 6, 1, 1, 2, 5, 4, 3, 1, 6, 1, 4, 1, 6, 1, 2, 2, 4, 8, 7,
1, 24, 1, 1, 2, 17, 1, 165, 1, 1, 16, 1, 2, 17, 9, 1, 3, 5, 2, 1, 3, 1, 2, 5, 1, 2, 3, 2, 4, 2, 22, 2, 4, 1, 1, 2, 4, 1, 3, 1,
2, 1, 131, 1, 2, 22, 5, 11, 1, 4, 14, 2, 2, 2, 10, 1, 2, 2, 1, 3, 1, 3, 1, 17, 1, 1, 2, 1, 3, 10, 1, 1, 1, 4, 1, 11, 1, 1, 1, 2,
69, 2, 1, 1, 1, 168, 3, 1, 1, 2, 4, 4, 1, 1, 53, 1, 15, 18, 6, 2, 3, 2, 1, 2, 4, 1, 23, 1, 4 |, 1, 1, 28, 2, 1, 1, 1, 1, 1, 1, 1,
2, 2, 3, 1, 2, 1, 3, 5, 3, 28, 1, 2, 2, 2, 2, 7, 4, 1, 1, 1, 1, 1, 6, 2, 3, 3, 47, 1, 1, 4, 2, 1, 1, 2, 1, 30, 4, 1, 1, 315,
1, 3, 30, 2, 1, 4, 1, 21, 1, 10, 1, 2, 1, 5, 9, 1, 26, 1, 1, 1, 4, 1, 5, 2, 457, 1, 1, 13, 9, 25, 2, 3, 1, 92, 1, 1, 3, 1, 2, 4,
158, 3, 4, 6, 2, 22, 1, 5, 1, 1, 1, 2, 4, 1, 1, 2, 6, 4, 2, 5, 2, 1, 1, 16, 47, 4, 1, 1, 2, 1, 2, 1, 1, 2, 3, 2, 3, 12, 7, 2,
2, 1, 5, 2, 1, 1, 1, 3, 1, 15, 1, 1, 1, 1, 1, 1, 7, 1, 101, 2, 1, 1, 5, 1, 1, 1, 1, 5, 1, 1, 1, 3, 4, 1, 9, 2, 1, 228, 1, 1,
3, 1, 2, 3, 7, 1, 1, 1, 12, 1, 2, 2, 10, 3, 2, 1, 14, 5, 2, 2, 1, 32, 1, 59, 2, 110, 1, 9, 1, 7, 9, 1, 7, 2, 1, 2, 1, 3, 5, 1,
1, 1, 1, 3, 8, 2, 2, 1, 2, 6, 1, 3, 7, 1, 7, 1, 1, 1, 10, 1, 1, 1, 1, 2, 1, 2, 1, 1, 1, 4, 3, 3, 18, 3, 3, 1, 1, 1, 1, 8,
1, 3, 1, 1, 6, 4, 9, 1, 3, 5, 1, 1, 3, 26, 38, 3, 6, 2, 2, 1, 1, 14, 1, 4, 1, 1, 3, 4, 1, 4, 1, 2, 2, 2, 1, 3, 15, 4, 1, 2,
1, 1, 6, 2, 1, 1, 1, 6, 11, 15, 1, 7, 3, 1, 1, 1, 3, 1, 1, 1, 11, 1, 1, 1, 5, 1, 5, 1, 1, 1, 9, 1, 1, 6, 25, 2, 2, 6, 2, 7,
4, 3, 1, 1, 1, 3, 2, 1, 6, 14, 2, 1, 1, 1, 2, 1, 6, 1, 17, 1, 1, 1, 18, 2, 1, 1, 1, 1, 1, 3, 1, 2, 1, 2, 34, 2, 3, 30, 1, 3,
2, 4, 1, 2, 1, 2, 1, 3, 1, 5, 1, 2, 1, 1, 1, 7, 1, 4, 6, 3, 5, 2, 2, 4, 2, 1, 10, 2, 1, 6, 1, 5, 1, 1, 11, 1, 6, 28, 1, 2,
9, 1, 2, 2, 1, 3, 1, 1, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 3, 4, 1, 1, 17, 1, 1, 4, 2, 7, 6, 6, 3, 4, 2, 14, 1, 6, 1, 2.

The CF expression of $\frac{t}{d}$ is as follows.

0, 8878, 1, 2, 14, 12, 1, 1, 1, 3, 18, 1, 54, 2, 7, 10, 1, 2, 4124, 1, 1, 1, 168, 22, 9, 3, 1, 1, 8, 1, 2, 1, 1, 4, 2, 2, 1, 1, 4, 3,
1, 1, 1, 9, 2, 1, 1, 1, 206, 1, 11, 1, 9, 4, 39, 3, 1, 86, 1, 2, 1, 6, 1, 1, 2, 5, 4, 3, 1, 6, 1, 4, 1, 6, 1, 2, 2, 4, 8, 7,
1, 24, 1, 1, 2, 17, 1, 165, 1, 1, 16, 1, 2, 17, 9, 1, 3, 5, 2, 1, 3, 1, 2, 5, 1, 2, 3, 2, 4, 2, 22, 2, 4, 1, 1, 2, 4, 1, 3, 1,
2, 1, 131, 1, 2, 22, 5, 11, 1, 4, 14, 2, 2, 2, 10, 1, 2, 2, 1, 3, 1, 3, 1, 17, 1, 1, 2, 1, 3, 10, 1, 1, 1, 4, 1, 11, 1, 1, 1, 2,
69, 2, 1, 1, 1, 168, 3, 1, 1, 2, 4, 4, 1, 1, 53, 1, 15, 18, 6, 2, 3, 2, 1, 2, 4, 1, 23, 1, 4.

Example 3. Refer to p, q of Example 1.

We consider $d > N^{\frac{1}{3}}$. Let $d =$

610336206651046900389953871563838676523222612329668538972313397403018544844267 4

868648018282242385291149523 (a 107 digit number). The corresponding e is

505408409935867461766002774357176472683450320736166597066744874470822439779184 13

692304683202474477009807257762032527139262517197626102515313556312250520329589 25

157211857561248864618212213360890463950145483676903110885853791616203089465206 09

520549715193549617689418034694789347338477123329904576457251773888159675951647 63 .

Now the value of $\frac{\frac{2N}{(2d+1)^2} - \frac{N}{N-A+1}}{\frac{N}{N-B} - \frac{N}{N-A+1}}$ is

277527825083860833403033559610727151722777672339402519575839704365461757777008 18

566756820931984066399162678299562973911555797293075406456976069250679242551518 89

806609322681839683564678529827434934279838966610424980004749617613593480863946 93

979893584592196652265784344928251903142309270176277560775333114134173734515 13,

which is smaller than e. The value of t is

135676363870987527877259750300665521941092949755408029431458162405448731998510 54

057524379767989315810471872.

The CF expression of $\frac{e}{N-\lceil \frac{3}{\sqrt{2}}\sqrt{N}\rceil+1}$ is as follows.

0, 4, 2, 163, 49, 1, 6, 10, 74, 1, 3, 2, 12, 1, 3, 1, 4, 1, 1, 1, 1, 2, 1, 1 ,4, 1, 2, 42, 21, 1, 1, 9, 2, 3, 1, 3, 1, 6, 1, 1,
2, 3, 19, 1, 2, 1, 2, 1, 7, 1, 35, 1, 11, 3, 14, 1, 2, 1, 3, 188, 1, 3, 5, 3, 1, 3, 1, 26, 2, 2, 1, 1, 1, 9, 1, 1, 3, 1, 4, 1,
1, 1, 3, 3, 1, 1, 1, 1, 2, 1, 1, 2, 4, 1, 11, 5, 2, 1, 7, 2, 1, 6, 3, 1, 3, 7, 1, 1, 1, 3, 1, 8, 9, 3, 5, 1, 4, 2, 1, 16,
1, 1, 1, 5, 2, 4, 1, 2, 1, 5, 1, 12, 2, 3, 2, 21, 2, 1, 6, 2, 3, 2, 1, 11, 1, 2, 1, 8, 1, 1, 2, 5, 1, 4, 4, 20, 2, 2, 22, 3,
2, 1, 2, 9, 6, 1, 1, 2, 3, 1, 1, 2, 1, 1, 15, 15, 1, 4, 1, 7, 1, 1, 1, 1, 1, 1, 5, 1, 2, 1, 1, 7, 7, 1, 2, 2, 7, 2, 11, 6,
1, 2, 223, 2, 4, 5, 1, 1, 9, 3, 3, 2, 1, 1, 5, 1, 3, 5, 1, 1, 1, 2, 26, 1, 1, 7, 10, 2, 1, 7, 4, 7, 1, 1, 5, 1, 4, 2, 2, 2,
3, 1, 5, 2, 1, 1, 2, 1, 1, 6, 6, 1, 10, 1, 33, 1, 6, 1, 3, 1, 2, 1, 2, 1, 1, 11, 3, 2, 8, 1, 29, 3, 2, 2, 36, 1, 5, 1, 2, 10,
9, 1, 4, 1, 9, 3, 1, 22, 4, 6, 10, 1, 1, 5, 10, 234, 1, 3, 13, 4, 9, 2, 1, 1, 2, 2, 1, 14, 1, 1, 2, 1, 1, 2, 5, 4, 1, 5, 1, 1,
4, 1, 62, 4, 8, 1, 8, 47, 10, 3, 2, 3, 7, 2, 2, 1, 2, 1, 1, 20, 1, 1, 1, 19, 440, 3, 3, 1, 6, 1, 2, 3, 2, 1, 3, 1, 1, 3, 1, 1,
48, 6, 2, 15, 21, 1, 4, 2, 3, 4, 234, 19, 50, 2, 18, 1, 3, 2, 2, 3, 3, 4, 1, 1, 5, 9, 7, 1, 3, 1, 1, 3, 1, 8, 1, 6, 2, 1, 24, 2,
14, 1, 6, 2, 2, 4, 6, 2, 1, 6, 7, 16, 3, 4, 8, 1, 1, 1, 3, 1, 2, 2, 1, 8, 1, 2, 2, 2, 1, 2, 1, 1, 4, 5, 1, 5, 1, 1, 14, 1,
1, 78, 2, 1, 2, 3, 3, 1, 1, 2, 4, 16, 1, 1, 1, 1, 1, 1, 1, 6, 2, 76, 2, 1, 2, 2, 2, 1, 1, 1, 1, 5, 2, 1, 1, 1, 1, 6, 1, 1,
1, 1, 5, 1, 29, 1, 40, 2, 1, 1, 7, 1, 3, 1, 4, 1, 5, 1, 4, 2, 1, 3, 1, 1, 2, 1, 15, 2, 9, 2, 1, 15, 1, 3, 2, 1, 1, 2, 9, 1,
2, 27, 2, 2, 1, 2, 3, 5, 3, 1, 4, 4, 1, 1, 1, 1, 7, 2, 5, 1, 1, 8, 2, 3, 1, 2, 1, 2, 1, 1, 1, 1, 1, 1, 36, 1, 1, 3, 16, 1, 1,
1, 1, 1, 6, 1, 1, 3, 1, 6, 2, 1, 1, 3, 5, 53, 3, 2, 2, 3, 4, 1, 3, 1, 1, 1, 1, 9, 1, 2, 9, 3, 1, 5, 1, 1, 1, 39, 3, 1, 1,
1, 2, 1, 18, 1, 1, 2, 1, 2, 4, 1, 1, 1, 1, 6, 1, 2, 1, 1, 4, 1, 1, 1, 4, 2, 30, 1, 3, 1, 18, 9, 1, 1, 1, 1, 31, 2, 44, 3117868, 11,
1, 3.

The CF expression of $\frac{t}{d}$ is as follows.

0, 4, 2, 163, 49, 1, 6, 10, 74, 1, 3, 2, 12, 1, 3, 1, 4, 1, 1, 1, 1, 2, 1, 1, 4, 1, 2, 42, 21, 1, 1, 9, 2, 3, 1, 3, 1, 6, 1, 1,
2, 3, 19, 1, 2, 1, 2, 1, 7, 1, 35, 1, 11, 3, 14, 1, 2, 1, 3, 188, 1, 3, 5, 3, 1, 3, 1, 26, 2, 2, 1, 1, 1, 9, 1, 1, 3, 1, 4, 1,
1, 1, 3, 3, 1, 1, 1, 1, 2, 1, 1, 2, 4, 1, 11, 5, 2, 1, 7, 2, 1, 6, 3, 1, 3, 7, 1, 1, 1, 3, 1, 8, 9, 3, 5, 1, 4, 2, 1, 16,

1, 1, 1, 5, 2, 4, 1, 2, 1, 5, 1, 12, 2, 3, 2, 21, 2, 1, 6, 2, 3, 2, 1, 11, 1, 2, 1, 8, 1, 1, 2, 5, 1, 4, 4, 20, 2, 2, 22, 3, 2, 1, 2, 9, 6, 1, 1, 2, 3, 1, 1, 2, 1, 1, 15, 15, 1, 4, 1, 7, 1, 1, 1, 1, 1, 1, 5, 1, 2, 1, 1, 7, 7, 1, 2, 2, 7, 2, 11, 6, 1, 2, 223, 2, 4, 5, 1, 1, 9, 3, 3, 2, 1, 1, 5, 2, 2, 10.

Note that the CF expression of $\frac{t}{d}$ could not be found (last three places do not match) in the CF expression of $\frac{e}{N-\lceil \frac{3}{\sqrt{2}}\sqrt{N}\rceil+1}$.

Deterministic Constructions of 21-Step Collisions for the SHA-2 Hash Family

Somitra Kumar Sanadhya* and Palash Sarkar

Applied Statistics Unit,
Indian Statistical Institute,
203, B.T. Road, Kolkata,
India 700108
somitra_r@isical.ac.in, palash@isical.ac.in

Abstract. Recently, at FSE '08, Nikolić and Biryukov introduced a new technique for analyzing SHA-2 round function. Building on their work, but using other differential paths, we construct two different deterministic attacks against 21-step SHA-2 hash family. Since the attacks are deterministic, they are actually combinatorial constructions of collisions. There are six free words in our first construction. This gives exactly 2^{192} different collisions for 21-step SHA-256 and exactly 2^{384} different collisions for 21-step SHA-512. The second construction has five free words. The best previous result, due to Nikolić and Biryukov, for finding collisions for 21-step SHA-256 holds with probability 2^{-19}. No results on 21-step SHA-512 are previously known. Further, we provide evidence that the Nikolić-Biryukov differential path is unlikely to yield 21-step collisions for SHA-512.

Keywords: SHA-2 family, cryptanalysis, reduced round attacks.

1 Introduction

At FSE '08, Nikolić and Biryukov [5] presented a 9-step local collision for the SHA-2 family. Using this local collision they presented an attack against step reduced SHA-256. All the prior research on SHA-2 family had considered local collisions which are valid for the linearized version of the round function of SHA-2 family. The first such linearized local collision was presented by Gilbert and Handschuh [2]. Later sixteen more linearized local collisions were shown by Sanadhya and Sarkar [6]. Linearized local collisions have been used in [3,4] and [7] to attack 18-step SHA-256.

The novelty of the Nikolić-Biryukov local collision lies in the fact that it is the first local collision which is valid for the actual round function of SHA-2 family. In contrast to the linearized local collisions which hold with probabilities around 2^{-39} or less, the Nikolić-Biryukov local collision holds with probability 1/3.

For the first time in the literature, the authors in [5] worked directly with modular differences for SHA-256 and obtained 20-step and 21-step collisions

* This author is supported by the Ministry of Information Technology, Govt. of India.

T.-C. Wu et al. (Eds.): ISC 2008, LNCS 5222, pp. 244–259, 2008.

for SHA-256 with probabilities 1/3 and 2^{-19} respectively. This was a marked improvement on the previous result of 19-step near collision differential path for SHA-256 presented by Mendel et al. [3].

Building on the work of Nikolić and Biryukov [5], but using a different 9-step non-linear local collision given by Sanadhya and Sarkar [8], we show two new 21-step attacks against SHA-2 family. Both these attacks are deterministic. For both these attacks, we provide algorithms to construct message pairs which collide after 21-steps of SHA-2 evaluations. Our two constructions have freedom of six and five message words respectively. This gives exactly 2^{192} different 21-step SHA-256 collisions and 2^{384} different 21-step SHA-512 collisions for the first construction. The corresponding values for the second construction are 2^{160} collisions for 21-step SHA-256 and 2^{320} collisions for 21-step SHA-512.

Nikolić and Biryukov had suggested that their 21-step attack against SHA-256 will also be valid for SHA-512 with the same probability, i.e. 2^{-19}. We show this to be not true. We provide evidence which points to the infeasibility of their attack against 21-step SHA-512. In showing this, we carefully analyze the differential behaviour of the linear function σ_1 used in the message expansion of SHA-512.

2 Notation

In this paper we use the following notation:

- Message words: $W_i \in \{0,1\}^n$, $W'_i \in \{0,1\}^n$ for any i. The word size n is 32 for SHA-256 and 64 for SHA-512.
- The colliding message pair: $\{W_0, W_1, W_2, \ldots W_{15}\}$ and $\{W'_0, W'_1, W'_2, \ldots W'_{15}\}$.
- The expanded message pair: $\{W_0, W_1, W_2, \ldots W_{r-1}\}$ and $\{W'_0, W'_1, W'_2, \ldots W'_{r-1}\}$. The number of steps r is 64 for SHA-256 and 80 for SHA-512.
- The internal registers for the two message pairs in step i: $\{a_i, \ldots, h_i\}$ and $\{a'_i, \ldots, h'_i\}$.
- $\mathrm{ROTR}^k(x)$: Right rotation of an n-bit quantity x by k bits.
- $\mathrm{SHR}^k(x)$: Right shift of an n-bit quantity x by k bits.
- $\oplus$: bitwise XOR.
- $+$: addition modulo 2^n.
- $-$: subtraction modulo 2^n.
- $\delta X = X' - X$ where X is an n-bit quantity.
- $\delta\Sigma_1(e_i) = \Sigma_1(e'_i) - \Sigma_1(e_i)$.
- $\delta\Sigma_0(a_i) = \Sigma_0(a'_i) - \Sigma_0(a_i)$.
- $\delta f^i_{MAJ}(x,y,z)$: Output difference of the f_{MAJ} function in step i when its inputs differ by x, y and z. That is, $\delta f^i_{MAJ}(x,y,z) = f_{MAJ}(a_i+x, b_i+y, c_i+z) - f_{MAJ}(a_i, b_i, c_i)$.
- $\delta f^i_{IF}(x,y,z)$: Output difference of the f_{IF} function in step i when its inputs differ by x, y and z. That is, $\delta f^i_{IF}(x,y,z) = f_{IF}(e_i + x, f_i + y, g_i + z) - f_{IF}(e_i, f_i, g_i)$.

3 The SHA-2 Hash Family and Collisions Attacks

3.1 The SHA-2 Hash Family

The SHA-2 hash function was standardized by NIST in 2002 [9]. There are 2 differently designed functions in this standard: the SHA-256 and SHA-512. In addition, the standard also specifies their truncated versions: the SHA-224 and SHA-384 respectively. The number in the name of the hash function refers to the length of message digest produced by that function. Next we describe SHA-256 and SHA-512 in detail.

Eight registers are used in the evaluation of SHA-2. The initial value in the registers is specified by an $8 \times n$ bit IV, n=32 for SHA-256 and 64 for SHA-512. In Step i, the 8 registers are updated from (a_{i-1}, b_{i-1}, c_{i-1}, d_{i-1}, e_{i-1}, f_{i-1}, g_{i-1}, h_{i-1}) to (a_i, b_i, c_i, d_i, e_i, f_i, g_i, h_i) according to the following equations:

$$\left.\begin{aligned} a_i &= \Sigma_0(a_{i-1}) + f_{MAJ}(a_{i-1}, b_{i-1}, c_{i-1}) + \Sigma_1(e_{i-1}) \\ &\quad + f_{IF}(e_{i-1}, f_{i-1}, g_{i-1}) + h_{i-1} + K_i + W_i \\ b_i &= a_{i-1} \\ c_i &= b_{i-1} \\ d_i &= c_{i-1} \\ e_i &= d_{i-1} + \Sigma_1(e_{i-1}) + f_{IF}(e_{i-1}, f_{i-1}, g_{i-1}) \\ &\quad + h_{i-1} + K_i + W_i \\ f_i &= e_{i-1} \\ g_i &= f_{i-1} \\ h_i &= g_{i-1} \end{aligned}\right\} \quad (1)$$

The initial register values $\{a_{-1}, b_{-1}, \ldots h_{-1}\}$ are specified by the IV. The f_{IF} and the f_{MAJ} are three variable boolean functions *If* and *Majority* respectively.

For SHA-256, the functions Σ_0 and Σ_1 are defined as:

$$\Sigma_0(x) = ROTR^2(x) \oplus ROTR^{13}(x) \oplus ROTR^{22}(x),$$
$$\Sigma_1(x) = ROTR^6(x) \oplus ROTR^{11}(x) \oplus ROTR^{25}(x).$$

For SHA-512, the corresponding functions are:

$$\Sigma_0(x) = ROTR^{28}(x) \oplus ROTR^{34}(x) \oplus ROTR^{39}(x),$$
$$\Sigma_1(x) = ROTR^{14}(x) \oplus ROTR^{18}(x) \oplus ROTR^{41}(x).$$

Round i uses an n-bit word W_i which is derived from the message and a constant word K_i. There are $r = 64$ steps in SHA-256 and 80 in SHA-512. The hash function operates on a 512-bit (resp. 1024-bit) block specified as 16 words of 32 (resp. 64) bits for SHA-256 (resp. SHA-512). Given the message words m_0, m_1, $\ldots m_{15}$, the W_i 's are computed using the equation:

$$W_i = \begin{cases} m_i & \text{for } 0 \le i \le 15; \\ \sigma_1(W_{i-2}) + W_{i-7} + \sigma_0(W_{i-15}) + W_{i-16} & \text{for } 16 \le i \le r. \end{cases} \quad (2)$$

For SHA-256, the functions σ_0 and σ_1 are defined as:

$$\sigma_0(x) = ROTR^7(x) \ \oplus ROTR^{18}(x) \oplus SHR^3(x),$$
$$\sigma_1(x) = ROTR^{17}(x) \oplus ROTR^{19}(x) \oplus SHR^{10}(x).$$

And for SHA-512, they are defined as:

$$\sigma_0(x) = ROTR^1(x) \oplus ROTR^8(x) \oplus SHR^7(x),$$
$$\sigma_1(x) = ROTR^{19}(x) \oplus ROTR^{61}(x) \oplus SHR^6(x).$$

The output hash value of a one block (512-bit for SHA-256 and 1024-bit for SHA-512) message is obtained by chaining the IV with the register values at the end of the final round as per the Merkle-Damgård construction. A similar strategy is used for multi-block messages, where the IV for next block is taken as the hash output of the previous block. For complete details of the SHA-2 family, see [9].

3.2 Collision Attacks Against the SHA-2 Hash Family

The aim of collision attacks against hash functions is to obtain two different messages which produce the same digest under that hash function. The hash functions use one word of the message in each step and process the message for multiple steps. Typically, an attacker introduces a small difference in one word of the message. This initial difference is called the "perturbation message difference" [1]. Next few message words are chosen to differ in such a manner that all the introduced differences cancel themselves with high probability. These later message word differences are called "correction differences".

Not all the message words used in different steps of the hash function are freely available to the attacker. Most of the hash designs have 16 words of freedom which is available in the first 16 steps of hash evaluation. Rest of the message words are computed on the basis of the first 16 words using "message expansion".

We present the nonlinear local collisions for the SHA-2 family next.

4 Nonlinear Local Collisions for SHA-2

Tables 1 and 2 show two 9-step local collision for the SHA-2 family. The first one is due to Nikolić and Biryukov [5] whereas the second one is due to Sanadhya and Sarkar [8]. In both the local collisions, the perturbation message difference is taken to be 1. Other message differences are later computed so that the desired differential path is obtained. In these tables, the registers $(a_{i-1}, \ldots, h_{i-1})$ and W_i are inputs to Step i of the hash evaluation and this step outputs the registers $(a_i, \ldots, h_i)$.

4.1 Conditions on the Differential Paths of Tables 1 and 2

For the Nikolić-Biryukov Local Collision [5]: For this local collision, δW_{i+1}, δW_{i+2} and δW_{i+3} are computed from the following equations:

$$\delta W_{i+1} = -1 - \delta f_{IF}^{i}(1,0,0) - \delta\Sigma_1(e_i), \tag{3}$$

$$\delta W_{i+2} = -\delta f_{IF}^{i+1}(-1,1,0) - \delta\Sigma_1(e_{i+1}), \tag{4}$$

Table 1. The Nikolić-Biryukov local collision [5]

Step i	δW_i	δa_i	δb_i	δc_i	δd_i	δe_i	δf_i	δg_i	δh_i
$i-1$	0	0	0	0	0	0	0	0	0
i	1	1	0	0	0	1	0	0	0
$i+1$	δW_{i+1}	0	1	0	0	-1	1	0	0
$i+2$	δW_{i+2}	0	0	1	0	0	-1	1	0
$i+3$	δW_{i+3}	0	0	0	1	0	0	-1	1
$i+4$	0	0	0	0	0	1	0	0	-1
$i+5$	0	0	0	0	0	0	1	0	0
$i+6$	0	0	0	0	0	0	0	1	0
$i+7$	0	0	0	0	0	0	0	0	1
$i+8$	-1	0	0	0	0	0	0	0	0

Table 2. The Sanadhya-Sarkar local collision [8]. We use this local collision in the present work.

Step i	δW_i	δa_i	δb_i	δc_i	δd_i	δe_i	δf_i	δg_i	δh_i
$i-1$	0	0	0	0	0	0	0	0	0
i	1	1	0	0	0	1	0	0	0
$i+1$	δW_{i+1}	0	1	0	0	-1	1	0	0
$i+2$	δW_{i+2}	0	0	1	0	-1	-1	1	0
$i+3$	δW_{i+3}	0	0	0	1	0	-1	-1	1
$i+4$	0	0	0	0	0	1	0	-1	-1
$i+5$	0	0	0	0	0	0	1	0	-1
$i+6$	0	0	0	0	0	0	0	1	0
$i+7$	δW_{i+7}	0	0	0	0	0	0	0	1
$i+8$	-1	0	0	0	0	0	0	0	0

$$\delta W_{i+3} = -\delta f_{IF}^{i+2}(0, -1, 1). \tag{5}$$

Intermediate registers need to satisfy the following conditions:

$$\begin{aligned} &a_{i-2} = a_{i-1} = a_{i+1} = a_{i+2},\ a_i = -1, \\ &e_{i+2} = e_{i+3},\ e_{i+4} = -1,\ e_{i+5} = 0,\ e_{i+6} = -1. \end{aligned} \tag{6}$$

In addition, an extra condition needs to be satisfied:

$$\delta f_{IF}^{i+3}(0, 0, -1) = -1. \tag{7}$$

All the conditions in (6) can be deterministically satisfied by choosing message words carefully, but the condition in (7) needs to be satisfied probabilistically. This causes the success probability of $1/3$ for this local collision. For details refer to [5]. Note that our notation and indexing of the steps is different from [5].

For the Sanadhya-Sarkar Local Collision [8]: For this local collision, δW_{i+1}, δW_{i+2} and δW_{i+3} are computed from the following equations:

$$\delta W_{i+1} = -1 - \delta f_{IF}^{i}(1,0,0) - \delta \Sigma_1(e_i), \tag{8}$$

$$\delta W_{i+2} = -1 - \delta f_{IF}^{i+1}(-1,1,0) - \delta \Sigma_1(e_{i+1}), \tag{9}$$

$$\delta W_{i+3} = -\delta f_{IF}^{i+2}(-1,-1,1) - \delta \Sigma_1(e_{i+2}), \tag{10}$$

$$\delta W_{i+7} = -\delta f_{IF}^{i+6}(0,0,1). \tag{11}$$

Intermediate registers need to satisfy the following conditions:

$$\begin{aligned} &a_{i-2} = a_{i-1} = a_i = -1,\ a_{i+1} = a_{i+2} = 0, \\ &e_{i+2} = 0,\ e_{i+3} = e_{i+4} = e_{i+5} = -1. \end{aligned} \tag{12}$$

We present two determintistic attacks in this work. We require $\delta W_{i+7} = 0$ in the first attack but not in the second. For having $\delta W_{i+7} = 0$, an extra condition $e_{i+6} = -1$ is required to be added to (12).

All the conditions in (12), with or without the condition $e_{i+6} = -1$, can be deterministically satisfied by choosing message words carefully. This ensures the success probability of 1 for this local collision. These conditions can be derived in the same way as in [5]. Refer to [8] for details on the derivation of these conditions and the method to satisfy them.

5 Deterministically Constructing 21-Step Collisions for the SHA-2 Family

In [5], a single local collision spanning from Step 6 to Step 14 is used and a 21-step collision for SHA-256 is obtained probabilistically. We use similar method for our attack but this time we choose a special case of the Sanadhya-Sarkar local collision in which $\delta W_{i+7} = 0$. As in [5], this local collision is also started from Step 6.

First 5 steps of message expansion for SHA-2 are:

$$\left.\begin{aligned} W_{16} &= \underline{\sigma_1(W_{14}) + W_9} + \sigma_0(W_1) + W_0, \\ W_{17} &= \sigma_1(W_{15}) + W_{10} + \sigma_0(W_2) + W_1, \\ W_{18} &= \underline{\sigma_1(W_{16})} + W_{11} + \sigma_0(W_3) + W_2, \\ W_{19} &= \underline{\sigma_1(W_{17})} + W_{12} + \sigma_0(W_4) + W_3, \\ W_{20} &= \underline{\sigma_1(W_{18})} + W_{13} + \sigma_0(W_5) + W_4. \end{aligned}\right\} \tag{13}$$

Since the chosen local collision has 4 consecutive zero message differentials within its span, we have $\delta W_i = 0$ for $i \in \{10, 11, 12, 13\}$. Further, this being the only local collision, messages outside the span of the local collision do not have

any difference. Thus, we also have $\delta W_i = 0$ for $i \in \{0, 1, 2, 3, 4, 5, 15\}$. Terms which *may have* non-zero differentials in the above equations are underlined.

All these zero differentials imply that if $\delta\sigma_1(W_{14}) + \delta W_9 = 0$ then the first 5 steps of the message expansion will not produce any difference, and we will have a 21-step collision. Since both W_{14} and W_9 are random, it can be expected that they will cancel the differences in this manner. Note that W_{14} is random but $\delta W_{14} = -1$ is fixed, whereas δW_9 is expected to be random.

Essentially, we need some δW_9 and W_{14} satisfying the following equation:

$$\sigma_1(W_{14}) - \sigma_1(W_{14} - 1) = \delta W_9. \tag{14}$$

Satisfying (14) will be easy if we can choose δW_9 to be an arbitrary value of our choice. In that case, we can first choose a random W_{14} and fix the value of δW_9 which satisfies (14). We now show that our attack indeed enables us to choose δW_9 and hence this condition can be satisfied deterministically.

Before describing our attack, we first discuss an important relationship between register values computed at each step of SHA-2.

5.1 Cross Dependence Equation

In the calculation of new register values at each step of the SHA-2 hash family, registers b, c and d are just copies of register a values of previous steps. Registers e and a are also related since most of the terms in their computation are common. Thus, we note that e_i can be computed solely from the register a values as shown below.

$$\begin{aligned} e_i &= d_{i-1} + \Sigma_1(e_{i-1}) + f_{IF}(e_{i-1}, f_{i-1}, g_{i-1}) + h_{i-1} + K_i + W_i \\ &= d_{i-1} + a_i - \Sigma_0(a_{i-1}) - f_{MAJ}(a_{i-1}, b_{i-1}, c_{i-1}) \\ &= a_{i-4} + a_i - \Sigma_0(a_{i-1}) - f_{MAJ}(a_{i-1}, a_{i-2}, a_{i-3}). \end{aligned} \tag{15}$$

This relationship between these two register values was not reported earlier. Equation 15 plays a crucial role in our attack. We call this equation the "Cross Dependence Equation".

The surprising thing to note in the Cross Dependence Equation is that we can control the value of e_i from a_{i-4}, a register value which was computed 4 steps earlier. We use this fact in constructing our attack.

5.2 Values of δW_9 for the Sanadhya-Sarkar Local Collision

Some register values are fixed for this local collision as given in (12). Recall that the local collision is started from step $i = 6$. Hence, the message difference δW_9 corresponds to δW_{i+3} from (5). This equation gives:

$$\begin{aligned} \delta W_{i+3} &= -\delta f_{IF}^{i+2}(-1, -1, 1) - \delta\Sigma_1(e_{i+2}) \\ &= -f_{IF}(e_{i+2} - 1, f_{i+2} - 1, g_{i+2} + 1) + f_{IF}(e_{i+2}, f_{i+2}, g_{i+2}) \\ &\quad -\Sigma_1(e_{i+2} - 1) + \Sigma_1(e_{i+2}) \\ &= -f_{IF}(e_{i+2} - 1, e_{i+1} - 1, e_i + 1) + f_{IF}(e_{i+2}, e_{i+1}, e_i) \\ &\quad -\Sigma_1(e_{i+2} - 1) + \Sigma_1(e_{i+2}) \end{aligned}$$

$$\begin{aligned}
&= -f_{IF}(0-1, e_{i+1}-1, e_i+1) + f_{IF}(0, e_{i+1}, e_i) \\
&\quad -\Sigma_1(0-1) + \Sigma_1(0) \\
&= -(e_{i+1}-1) + e_i - (-1) + 0 \\
&= -e_{i+1} + e_i + 2 \;=\; -e_7 + e_6 + 2.
\end{aligned}$$

We also know from the "Cross Dependence Equation" (15) that:

$$\begin{aligned}
e_6 &= a_2 + a_6 - \Sigma_0(a_5) - f_{MAJ}(a_5, a_4, a_3), \\
e_7 &= a_3 + a_7 - \Sigma_0(a_6) - f_{MAJ}(a_6, a_5, a_4).
\end{aligned}$$

Substituting the values of registers from (12) with $i = 6$, we get:

$$\begin{aligned}
e_6 &= a_2 + a_6 - \Sigma_0(a_5) - f_{MAJ}(a_5, a_4, a_3) \\
&= a_2 + (-1) - \Sigma_0(-1) - f_{MAJ}(-1, -1, a_3) \\
&= a_2 + (-1) - (-1) - (-1) \;=\; a_2 + 1, \\
e_7 &= a_3 + a_7 - \Sigma_0(a_6) - f_{MAJ}(a_6, a_5, a_4) \\
&= a_3 + 0 - \Sigma_0(-1) - f_{MAJ}(-1, -1, -1) \\
&= a_3 + 0 - (-1) - (-1) \;=\; a_3 + 2.
\end{aligned}$$

This gives,

$$\delta W_9 = -(a_3 + 2) + (a_2 + 1) + 2 \;=\; -a_3 + a_2 + 1. \tag{16}$$

This value of δW_9 for the local collision of Table 2 depends on the registers a_3 and a_2 only. Register value a_2 will be available in Step 3 of hash evaluation and register value a_3 will be produced in this step using W_3. We can therefore choose W_3 suitably to fix the difference δW_9 to any arbitrary desired value. *It is startling to note that δW_9 can be fixed to any desired value by choosing a message word 6 steps earlier.* This is a consequence of the Cross Dependence Equation.

5.3 Obtaining 21-Step Collisions

Recall that (11) is used at Step i of the hash evaluation. Registers $(a_{i-1}, b_{i-1}, \ldots, h_{i-1})$ are available at this step and the output register a_i or e_i can be controlled by selecting W_i suitably. For instance, if we wish to make a_i to be zero, then we can calculate the suitable value of W_i from (11) which will make this happen.

The algorithm to obtain message pairs leading to deterministic 21-step collisions for SHA-2 family in described in Section B. Messages colliding for 21-step SHA-512 and 21-step SHA-256 obtaied using this algorithm are shown in Table 3.

6 Another Construction for Deterministic 21-Step Collisions in SHA-2

This time we choose to span a single instance of the Sanadhya-Sarkar local collision from Step 7 to Step 15. We have that $\delta W_7 = 1$ and $\delta W_{15} = -1$.

Further, $\delta W_8, \delta W_9, \delta W_{10}$ and δW_{14} are determined from (8), (9), (10) and (11) respectively.

If we can ensure that the first two steps of message expansion do not produce any message difference then (13) implies that there will be no message difference in the next three steps as well. This will give another 21-step collision for SHA-2 family. The conditions needed to be satisfied for handling first two steps of message expansion as desired are:

$$\sigma_1(W_{14}) + W_9 = \sigma_1(W'_{14}) + W'_9, \tag{17}$$

$$\sigma_1(W_{15}) + W_{10} = \sigma_1(W'_{15}) + W'_{10}. \tag{18}$$

We know that $\delta W_{15} = W'_{15} - W_{15} = -1$. Further, $\delta W_{10} = W'_{10} - W_{10}$ for this case is similar to the δW_9 of Section 5.2 and can be set to any arbitrarily chosen value. Therefore (18) can be deterministically satisfied using a method similar to that described in Section 5.2. We now carefully examine (17) which is the only condition left to be satisfied.

Equation 17 can be written as:

$$\sigma_1(W_{14}) - \sigma_1(W_{14} + \delta W_{14}) = \delta W_9. \tag{19}$$

The use of the local collision of Table 2 causes some register values to be set as per (12). In particular, we have that $e_{12} = e_{11} = -1$. From (11), we get:

$$\begin{aligned}
\delta W_{14} &= -\delta f_{IF}^{13}(0,0,1) \\
&= -f_{IF}(e_{13}, f_{13}, g_{13}+1) + f_{IF}(e_{13}, f_{13}, g_{13}) \\
&= -f_{IF}(e_{13}, e_{12}, e_{11}+1) + f_{IF}(e_{13}, e_{12}, e_{11}) \\
&= -f_{IF}(e_{13}, -1, 0) + f_{IF}(e_{13}, -1, -1) \\
&= -e_{13} - 1.
\end{aligned}$$

The above expression for δW_{14} depends only on e_{13}. Register e_{13} can be set to any desired value by proper choice of the free message word W_{13}. Hence we can have any desired value of δW_{14} for our collision. Coming back to (19), we observe that we have no control over δW_9. However, for any choice of δW_9, we can always find a solution to (19). For example, if we write $W_{14} = A$ and $\delta W_{14} = B - A$, then we need to solve the equation $\sigma_1(A) - \sigma_1(B) = \delta W_9$. There are many solutions to this equation. A particular solution is obtained by choosing $A = \sigma_1^{-1}(\delta W_9)$ and $B = 0$. This solution corresponds to $W_{14} = \sigma_1^{-1}(\delta W_9)$ and $\delta W_{14} = -W_{14}$.

The particular solution suggested above will work only if we can invert the 32×32 bit map σ_1 for SHA-256 and the 64×64 bit map σ_1 for SHA-512. We note that the map σ_1 is a linear function. Therefore $\sigma_1(x)$ can be expressed as multiplication of a matrix with x. For both these hash functions, the corresponding matrix is of full rank. Therefore σ_1 is invertible and our attack succeeds with probability one. The algorithm to obtain message pairs colliding for 21-step SHA-2 family using this construction is similar to our first construction given in Table 5. Messages colliding for 21-step SHA-512 and 21-step SHA-256 obtaied using our second algorithm are shown in Table 4.

7 Infeasibility of 21-Step SHA-512 Collision Using Nikolić-Biryukov Local Collision

For this local collision, we first show the difficulty of finding values of δW_9 and $\delta\sigma_1(W_{14})$ which are of the same order of magnitude. We note that δW_9 is biased towards values of small magnitude in this case. In contrast, $\sigma_1(W_{14}) - \sigma_1(W_{14}-1)$ for SHA-512 is biased towards large magnitudes for random values of W_{14}. This makes it unlikely to achieve equality of the two terms as required in (14). Now we provide concrete proofs for these facts.

7.1 Magnitude of δW_9 Values for the Nikolić-Biryukov Local Collision

We first state and prove some propositions which help us understand the bias of δW_9 in this case. Later we prove a lemma regarding the magnitude of δW_9. In the discussion that follows, we use X_i to denote the i^{th} bit of a 64-bit quantity X. We also use the convention that the index of the least significant bit is 0.

Proposition 1. *$Pr[P_j \neq (P+1)_j] = 1/2^j$, where the probability is taken over random P.*

Proof. The necessary and sufficient condition for the j^{th} bit of P and $P+1$ to differ is that all the bits from 0 to $(j-1)$ in P are 1. This happens with probability $1/2^j$, hence proved. □

Proposition 2. *If two numbers X and Y are such that $X_i \neq Y_i$ and $X_{i-1} = Y_{i-1}$, then $|X - Y| \geq 2^{i-1} + 1$.*

Proof. Without loss of generality, suppose $X_i = 1$ and $Y_i = 0$. Let $Z = X - Y$. If $Z_i = 1$, then clearly $|Z| \geq 2^i$ and we are done.

So, suppose $Z_i = 0$ and consider the process of binary subtraction of Y from X to obtain Z. Since $X_i = 1$ and $Y_i = 0$, the result $Z_i = 0$ can happen only if the subtraction of $Y_{i-1}Y_{i-2}\ldots Y_0$ from $X_{i-1}X_{i-2}\ldots X_0$ produces a carry. But since $X_{i-1} = Y_{i-1}$, this implies the following two things.

1. $Z_{i-1} = 1$.
2. The subtraction of $Y_{i-2}Y_{i-3}\ldots Y_0$ from $X_{i-2}X_{i-3}\ldots X_0$ produces a carry.

The second point implies that at least one bit of $Z_{i-2}Z_{i-3}\ldots Z_0$ must be 1. This together with the first point $Z_{i-1} = 1$ implies that $|Z| \geq 2^{i-1} + 1$. Hence proved. □

Next we prove that the probability that the absolute value of δW_9, when using Nikolić-Biryukov local collision, is larger than 2^j is bounded above by $1/2^{j-1}$.

Lemma 1. *If the Nikolić-Biryukov local collision is started at Step 6, then*

$$Pr[|\delta W_9| \geq 2^j] \; < \; 1/2^{j-1}.$$

Proof. Since the local collision is started from step $i = 6$, the message difference δW_9 corresponds to δW_{i+3} from (5). This equation gives:

$$\begin{aligned}\delta W_9 &= -\delta f_{IF}^8(0,-1,1) - \delta\Sigma_1(e_8) \\ &= -f_{IF}(e_8, f_8 - 1, g_8 + 1) + f_{IF}(e_8, f_8, g_8) - 0 \\ &= -f_{IF}(e_8, e_7 - 1, e_6 + 1) + f_{IF}(e_8, e_7, e_6). \end{aligned} \tag{20}$$

The two f_{IF} terms in the computation above have the same first argument e_8. The second and the third arguments have a modular difference of ± 1. If the j^{th} bit of e_8 is 1 then the two f_{IF} functions will select the corresponding bit from the middle argument, else from the third argument.

Let $A = f_{IF}(e_8, e_7 - 1, e_6 + 1)$ and $B = f_{IF}(e_8, e_7, e_6)$. Further, let P_i be the event that $A_i \neq B_i$. The event $\delta W_9 \geq 2^j$ can happen if and only if at least one of the bits $j, j+1, \ldots 63$ of δW_9 is 1, i.e., if and only if at least one of the events $P_j, P_{j+1}, \ldots P_{63}$ holds.

Now we are ready to bound the probability of the required event. In the fourth step below, we use (20) and the fact that $f_{IF}(a,b,c) = b$ if $a = 1$ and $f_{IF}(a,b,c) = c$ if $a = 0$.

$$\begin{aligned} Pr[\delta W_9 \geq 2^j] &= Pr\left[\bigcup_{i \geq j} P_i\right] \\ &\leq \sum_{i \geq j} Pr[P_i] \\ &= \sum_{i \geq j} (Pr[(e_8)_i = 0] \times \\ &\qquad Pr[P_i | ((e_8)_i = 0)] + Pr[(e_8)_i = 1] \cdot Pr[P_i | ((e_8)_i = 1)]) \\ &= \sum_{i \geq j} \left(\frac{1}{2} \cdot Pr[(e_6 + 1)_i \neq e_6] + \frac{1}{2} \cdot Pr[(e_7 - 1)_i \neq e_7] \right) \\ &= \frac{1}{2} \cdot \sum_{i \geq j} \left(\frac{1}{2^i} + \frac{1}{2^i} \right) \quad \text{(Using Proposition 1)} \\ &< \frac{1}{2^{j-1}}. \end{aligned}$$

This proves the Lemma. □

7.2 Magnitude of $\sigma_1(W) - \sigma_1(W-1)$ Values for SHA-512

We now look at the distribution of values of $\sigma_1(W) - \sigma_1(W-1)$ for random choices of W. By using the combinatorial structure of the function σ_1 and partial search using a computer program, it is possible to prove the following lemma.

Lemma 2. *For the function σ_1 used in SHA-512,*

$$|\sigma_1(W) - \sigma_1(W-1)| \geq (2^{42} + 2^{39} + 2^{38} + 2^{36} - 2^3).$$

For proof of this lemma, refer to Section A.

7.3 Infeasibility of the 21-Step Collision Using NB Differential Path

From Lemma 1, we get that the probability that a value of δW_9 produced when using this local collision is larger than 2^{42} is less than $1/2^{41}$. That is, on average one will require 2^{41} or more attempts with the differential path to get a single value of δW_9 which is larger than 2^{42}. On the other hand, Lemma 2 shows that all the values of $\sigma_1(W_{14}) - \sigma_1(W_{14} - 1)$ will be larger than 2^{42}.

In addition, the proof of Lemma 2 makes it clear that the term $\sigma_1(W_{14}) - \sigma_1(W_{14} - 1)$ has a strong structure which is far from random. Our experiments support this view further. For instance, we experimentally observed that this difference of σ_1 terms has a large trail of zero bits or a large trail of one bits in the middle. The number of ones or zeros in the continuous sequence are almost always between 20 to 35. Further, there are many 64-bit words which occur repeatedly as the value of this term for different choices of W_{14}. Also, some values are never achieved. It is not clear whether it is possible to achieve such a strongly structured pattern in δW_9 and satisfy (14) with the use of the Nikolić-Biryukov local collision for SHA-512.

Note that this local collision succeeds for the SHA-256 case because the choice of the two rotation values used in the σ_1 function for SHA-256 are not far apart. This causes most of the bits to overlap over nearby bits and the bias of the term $\sigma_1(W_{14}) - \sigma_1(W_{14} - 1)$ is not as skewed as in the case of SHA-512.

8 Some Concluding Remarks

In this work we have presented two deterministic constructions for 21-step collisions for SHA-2 hash family. This improves on the recent attack by Nikolić and Biryukov at FSE '08. We have also analyzed the linear function σ_1 for SHA-512 and shown that its differential behaviour is quite peculiar. We then analyzed the 21-step collision of Nikolić and Biryukov and have shown that it is not likely to succeed for SHA-512. We hope this work will help understand the behaviour of SHA-2 better and will help in developing future attacks on the SHA-2 family.

References

1. Chabaud, F., Joux, A.: Differential Collisions in SHA-0. In: Krawczyk, H. (ed.) CRYPTO 1998. LNCS, vol. 1462, pp. 56–71. Springer, Heidelberg (1998)
2. Gilbert, H., Handschuh, H.: Security Analysis of SHA-256 and Sisters. In: Matsui, M., Zuccherato, R.J. (eds.) SAC 2003. LNCS, vol. 3006, pp. 175–193. Springer, Heidelberg (2003)
3. Mendel, F., Pramstaller, N., Rechberger, C., Rijmen, V.: Analysis of Step-Reduced SHA-256. In: Robshaw, M. (ed.) FSE 2006. LNCS, vol. 4047, pp. 126–143. Springer, Heidelberg (2006)
4. Mendel, F., Pramstaller, N., Rechberger, C., Rijmen, V.: Analysis of Step-Reduced SHA-256. Cryptology eprint Archive, (March 2008), http://eprint.iacr.org/2008/130

5. Nikolić, I., Biryukov, A.: Collisions for Step-Reduced SHA-256. In: Nyberg, K. (ed.) FSE 2008. LNCS, vol. 5086, pp. 1–16. Springer, Heidelberg (2008)
6. Sanadhya, S.K., Sarkar, P.: New Local Collisions for the SHA-2 Hash Family. In: Nam, K.-H., Rhee, G. (eds.) ICISC 2007. LNCS, vol. 4817, pp. 193–205. Springer, Heidelberg (2007)
7. Sanadhya, S.K., Sarkar, P.: Attacking Reduced Round SHA-256. In: Bellovin, S., Gennaro, R., Keromytis, A.D., Yung, M. (eds.) ACNS 2008. LNCS, vol. 5037, Springer, Heidelberg (2008)
8. Sanadhya, S.K., Sarkar, P.: Non-Linear Reduced Round Attacks Against SHA-2 Hash family. In: Mu, Y., Susilo, W. (eds.) ACISP 2008. LNCS, vol. 5107. Springer, Heidelberg (2008)
9. Secure Hash Standard. Federal Information Processing Standard Publication 180-2. U.S. Department of Commerce, National Institute of Standards and Technology(NIST) (2002), http://csrc.nist.gov/publications/fips/fips180-2/fips180-2withchangenotice.pdf

A Proof of Lemma 2

Proof. The function σ_1 is defined for SHA-512 as:

$$\sigma_1(W) = ROTR^{19}(W) \oplus ROTR^{61}(W) \oplus SHR^6(W). \tag{21}$$

Let the 64-bit word W be specified as $(w_{63}, w_{62}, \ldots, w_1, w_0)$ where w_0 is the least significant bit of W. Then $\sigma_1(W)$ can be expressed as bit-wise XOR of three quantities having bit pattern shown below.

Bit Index	63	62	...	58	57	...	45	44	...	0
$ROTR^{19}$	w_{18}	w_{17}	...	w_{13}	w_{12}	...	w_0	w_{63}	...	w_{19}
$ROTR^{61}$	w_{60}	w_{59}	...	w_{55}	w_{54}	...	w_{42}	w_{41}	...	w_{61}
SHR^6	0	0	...	0	w_{63}	...	w_{51}	w_{50}	...	w_6

Let $W' = W - 1$. Then similar structure for $\sigma_1(W')$ can also be visualized. We are interested in the magnitude of $\sigma_1(W) - \sigma_1(W')$.

Let j be the least index such j^{th} bit of W is 1. That is, $w_j = 1$ and $w_i = 0$ for all $i \leq j - 1$. Then we have, $w_i \neq w'_i$ for $i \leq j$ and $w_i = w'_i$ for $i > j$. Now we consider two cases for j.

Case 1: $0 \leq j \leq 40$.

In this case, we have that $w_i = w'_i$ for $i = 63, 51, 50, 42, 41$ and $w_0 \neq w'_0$. From the structure of $\sigma_1(W)$ and $\sigma_1(W')$, we note that their 45^{th} bits will be unequal but their 44^{th} bits will be equal. Using Proposition 2 we get, $|\sigma_1(W) - \sigma_1(W')| \geq 2^{44} + 1$.

Case 2: $j \geq 41$.

We need to consider the individual cases $j = 41, 42, \ldots 63$ here. Consider the case $j = 41$ first. In this case, we know the exact bit pattern in W and W' up to 41 bits. Only the high order bits from 42 to 63 are unknown in these two quantities. We also know that these high order bits are the same in W and W'. Since these are only 22 bits, we can exhaustively search this space and compute the value $|\sigma_1(W) - \sigma_1(W')|$ for the case $j = 41$. As j is increased, the same idea can be used with even smaller search space. The size of the complete search space is $1 + 2 + 2^2 + \ldots + 2^{22} = 2^{23} - 1$. A C program running on an ordinary PC takes a fraction of a second to traverse this space.

Using exhaustive search, we found the minimum vale of $|\sigma_1(W) - \sigma_1(W - 1)|$ to be `000004cfffffff8` which occurred for $j = 42$. This value is equal to $(2^{42} + 2^{39} + 2^{38} + 2^{36} - 2^3)$.

We have left one particular case of W undiscussed. This is the special case when all the bits in W are zero. In this case, we can compute the difference directly since $\sigma_1(0) = 0$ and $\sigma_1(-1) = 1 + 2 + \ldots + 2^{57}$. Thus, we have the difference $= 2^{58} - 1$.

Combining all the cases, the Lemma is proved. □

B Algorithm for Obtaining 21-Step SHA-2 Collisions

The step by step construction for the first attack described in Section 5 is now presented. We define two functions which return the required message word W_i to set the register value a_i or e_i to desired values, say `desired_a` and `desired_e`, at Step i. Equation 1 provides the definitions of these two functions.

1. `W_to_set_register_A(Step` i`, desired_a, Current State` $\{a_{i-1}, b_{i-1}, \ldots, h_{i-1}\})$ `:=` $($`desired_a` $-\Sigma_0(a_{i-1}) - f_{MAJ}(a_{i-1}, b_{i-1}, c_{i-1}) - \Sigma_1(e_{i-1}) - f_{IF}(e_{i-1}, f_{i-1}, g_{i-1}) - h_{i-1} - K_i)$
2. `W_to_set_register_E(Step` i`, desired_e, Current State` $\{a_{i-1}, b_{i-1}, \ldots, h_{i-1}\})$`:=` $($`desired_e` $- d_{i-1} - \Sigma_1(e_{i-1}) - f_{IF}(e_{i-1}, f_{i-1}, g_{i-1}) - h_{i-1} - K_i)$

The actual construction for our first attack is described in Table 5. Note that all the conditions for the differential path of Table 2 are satisfied deterministically. The extra condition of (14) required for ensuring that first five steps of the message expansion do not produce any difference is also satisfied deterministically. Therefore, this procedure gives deterministic 21-step collisions for all hash functions of the SHA-2 family.

C Colliding Message Pairs

Table 3. Colliding message pair for 21-step SHA-512 and 21-step SHA-256 with standard IV. These messages have been generated using our first construction. The six free words are taken to be all zero in the first message.

W_1	0-3	0	0	0	f31b03afd93b0f6d
	4-7	8308c99162dbe6fe	739cde771413fca7	49404e11d70bcb89	b913ae32c4c3736f
	8-11	e284a20a666a735b	573c19c84c7c8968	a28b08dc36fe38d0	6db010d6afe33679
	12-15	8d41a28b0d847692	0	0	0
W_2	0-3	0	0	0	f31b03afd93b0f6d
	4-7	8308c99162dbe6fe	739cde771413fca7	49404e11d70bcb8a	b907ee32c543736f
	8-11	e290620a65ea735b	533c19c84c7c8969	a28b08dc36fe38d0	6db010d6afe33679
	12-15	8d41a28b0d847692	0	ffffffffffffffff	0

W_1	0-7	0	0	0	e9f8ba95	534d9ceb	c6ceba8e	3b9d659d	e5eb4a91
	8-15	93bc685a	76a3426e	e7f10e03	b6d615fd	8d41a28d	0	0	0
W_2	0-7	0	0	0	e9f8ba95	534d9ceb	c6ceba8e	3b9d659e	ea0b4a10
	8-15	8f9c68d8	7663426f	e7f10e03	b6d615fd	8d41a28d	0	ffffffff	0

Table 4. Colliding message pair for 21-step SHA-512 and 21-step SHA-256 with standard IV. These messages have been generated using our second construction. The five free words are taken to be all zero in the first message.

W_1	0-3	0	0	0	0
	4-7	33288419c029d472	bba39f8b68732f94	bb74290355d1ceb0	f335474a1b64b2c1
	8-11	d1fdacca89083b02	700713c821d7108c	a68b08dc36fe38cf	60cb1486891e45e7
	12-15	3f1934d56ca27159	734a1a4b3d4c54bb	f428cc497862eb6b	0
W_2	0-3	0	0	0	0
	4-7	33288419c029d472	bba39f8b68732f94	bb74290355d1ceb0	f335474a1b64b2c2
	8-11	d201ecca89883b01	7002d3c82157108a	a28b08dc36fe38d0	60cb1486891e45e7
	12-15	3f1934d56ca27159	734a1a4b3d4c54bb	0	ffffffffffffffff

W_1	0-7	0	0	0	0	c1c4ef2c	c6276387	780cfcca	43a08b03
	8-15	d51654ea	76eb0773	e8310e02	a0ced093	82dcf0e1	38812f86	b95fe183	0
W_2	0-7	0	0	0	0	c1c4ef2c	c6276387	780cfcca	43a08b04
	8-15	c8365867	83cb03ef	e7f10e03	a0ced093	82dcf0e1	38812f86	0	ffffffff

Table 5. Algorithm to obtain message pairs leading to deterministic collisions for 21-step SHA-2. This corresponds to our first attack described in Section 5.

external `W_to_set_register_A(Step` i`, desired_a, Current State` $\{a_{i-1}, b_{i-1}, \ldots, h_{i-1}\}$`)`: Returns the required message W_i to be used in step i so that a_i is set to the given value.

external `W_to_set_register_E(Step` i`, desired_e, Current State` $\{a_{i-1}, b_{i-1}, \ldots, h_{i-1}\}$`)`: Returns the required message W_i to be used in step i so that e_i is set to the given value.

First Message words:

1. Select W_0, W_1, W_2, W_{13}, W_{14} and W_{15} randomly.
2. Compute the required value of $\delta W_9 = \sigma_1(W_{14}) - \sigma_1(W_{14} - 1)$. (Refer (14))
3. Run Steps 0, 1 and 2 of hash evaluation to define $\{a_2, b_2, \ldots h_2\}$.
4. Choose W_3 = `W_to_set_register_A(3,` $a_2 + 1 - \delta W_9$`,` $\{a_2, b_2, \ldots h_2\}$`)`.
5. Run Step 3 of hash evaluation to define $\{a_3, b_3, \ldots h_3\}$.
6. Choose W_4 = `W_to_set_register_A(4, -1,` $\{a_3, b_3, \ldots h_3\}$`)`.
7. Run Step 4 of hash evaluation to define $\{a_4, b_4, \ldots h_4\}$.
8. Choose W_5 = `W_to_set_register_A(5, -1,` $\{a_4, b_4, \ldots h_4\}$`)`.
9. Run Step 5 of hash evaluation to define $\{a_5, b_5, \ldots h_5\}$.
10. Choose W_6 = `W_to_set_register_A(6, -1,` $\{a_5, b_5, \ldots h_5\}$`)`.
11. Run Step 6 of hash evaluation to define $\{a_6, b_6, \ldots h_6\}$.
12. Choose W_7 = `W_to_set_register_A(7, 0,` $\{a_6, b_6, \ldots h_6\}$`)`.
13. Run Step 7 of hash evaluation to define $\{a_7, b_7, \ldots h_7\}$.
14. Choose W_8 = `W_to_set_register_A(8, 0,` $\{a_7, b_7, \ldots h_7\}$`)`.
15. Run Step 8 of hash evaluation to define $\{a_8, b_8, \ldots h_8\}$.
16. Choose W_9 = `W_to_set_register_E(9, -1,` $\{a_8, b_8, \ldots h_8\}$`)`.
17. Run Step 9 of hash evaluation to define $\{a_9, b_9, \ldots h_9\}$.
18. Choose W_{10} = `W_to_set_register_E(10, -1,` $\{a_9, b_9, \ldots h_9\}$`)`.
19. Run Step 10 of hash evaluation to define $\{a_{10}, b_{10}, \ldots h_{10}\}$.
20. Choose W_{11} = `W_to_set_register_E(11, -1,` $\{a_{10}, b_{10}, \ldots h_{10}\}$`)`.
21. Run Step 11 of hash evaluation to define $\{a_{11}, b_{11}, \ldots h_{11}\}$.
22. Choose W_{12} = `W_to_set_register_E(12, -1,` $\{a_{11}, b_{11}, \ldots h_{11}\}$`)`.

Second message words:

23. Define $\delta W_i = 0$ for $i \in \{0, 1, 2, 3, 4, 5, 10, 11, 12, 13, 15\}$, $\delta W_6 = 1$ and $\delta W_{14} = -1$.
24. Define $\delta W_7 = -1 - f_{IF}(e_6 + 1, f_6, g_6) + f_{IF}(e_6, f_6, g_6) - \Sigma_1(e_6 + 1) + \Sigma_1(e_6)$. (Refer (8))
25. Define $\delta W_8 = -1 - f_{IF}(e_7 - 1, f_7 + 1, g_7) + f_{IF}(e_7, f_7, g_7) - \Sigma_1(e_7 - 1) + \Sigma_1(e_7)$. (Refer (9))
26. Define $\delta W_9 = -f_{IF}(e_8 - 1, f_8 - 1, g_8 + 1) + f_{IF}(e_8, f_8, g_8) - \Sigma_1(e_8 - 1) + \Sigma_1(e_8)$. (Refer (10))
27. Compute $W'_i = W_i + \delta W_i$ for $0 \leq i \leq 15$.

Similar algorithm for the attack described in Section 6 can be constructed. We do not provide the second algorithm due to space restrictions.

Proxy Re-signatures in the Standard Model

Sherman S.M. Chow[1] and Raphael C.-W. Phan[2,⋆]

[1] Department of Computer Science
Courant Institute of Mathematical Sciences
New York University, NY 10012, USA
schow@cs.nyu.edu
[2] Electronic & Electrical Engineering
Loughborough University
LE11 3TU, United Kingdom
r.phan@lboro.ac.uk

Abstract. This paper studies proxy re-signature schemes. We first classify the expected security notions for proxy re-signature schemes with different properties. We then show how to attack on a recently proposed bidirectional scheme that is purported to be secure without random oracles, and discuss the flaw in their proof. Next, we show how to design a generic unidirectional proxy re-signature scheme using a new primitive called homomorphic compartment signature as the building block. We give a concrete instantiation which yields the first known unidirectional proxy re-signature scheme which is proven secure under standard assumption in the standard model. We also discuss how to incorporate the concept of forward-security into the proxy re-signature paradigm, such that the signing and the transformation are both time-limited.

Keywords: Proxy re-signature, compartment signature, standard model.

1 Introduction

Proxy re-cryptography is about delegating transformation rights of cryptographic objects to a semi-trusted proxy, such that a cryptographic task which can only be completed by a delegator now becomes a task that can only be completed by a delegatee. This idea was introduced by Blaze, Bleumer, and Strauss [2]. For *proxy re-signature*, signatures signed by a delegator can be transformed into ones signed by a delegatee, without allowing the proxy to sign on any other messages.

1.1 Applications

Proxy re-signatures have many interesting applications [1,2]. One of which is about public-key certificates management. Public-key certificates issued by certification authorities are often deployed in e-commerce infrastructure to allow

⋆ Work done while the author was with the Laboratoire de sécurité et de cryptographie (LASEC), Ecole Polytechnique Fédérale de Lausanne (EPFL), Switzerland.

T.-C. Wu et al. (Eds.): ISC 2008, LNCS 5222, pp. 260–276, 2008.

validating online transactions. Proxy re-signatures ease the certificates deployment by transforming signatures of a certification authority's new public keys into signatures that can be verified with public keys already certified in existing certificates. Generalizing, this leads to transparent cross-certification between different certification authorities, i.e. certificates from one authority can be converted into certificates from the others.

Proxy re-signatures aid in the usage of machine readable travel documents like e-passports. From one point to another the signature within the passport can be transformed, beginning with the issuer's certification on the passport holder's identity, through the different check points, each time requiring only one transformable signature to be kept within the passport. Generalizing, it can also trace the path taken by a travelling salesperson on business trips.

Yet another use of proxy re-signatures is in the generation of anonymous group signatures, for instance by transforming any signature by an employee to one verifiable under the corporate's public key. This conceals the unique identity of the employee to prevent information leakage says on the company's corporate structure or employment profile, while still allowing for internal auditing.

1.2 Our Contributions

1. We build on the security notions for proxy re-signatures of Ateniese and Hohenberger [1], and discuss what to expect for schemes with different properties including private proxy, non-interactivity, transparency, transitivity and bidirectional properties. While their definition is general enough, not much discussion is made on which notion should be considered for different combination of properties and why a certain property is ensured.
2. We show how to attack on a bidirectional proxy re-signature scheme recently proposed by Shao *et al.* [10], that came with a security proof without random oracles. We pinpoint the reason accounting for the insecurity and suggest how the attacks could be prevented.
3. We generalize the concept of hierarchical signatures to the notion of (homomorphic) compartment signatures. Using this new primitive as a building block, we show how to design a generic proxy re-signature scheme.

Table 1. Comparison of Properties with Existing Proxy Re-Signature Schemes

Property / Scheme	BBS [2]	S_{bi} [1]	S_{uni} [1]	S^*_{uni} [1]	S_{mb} [10]	Proposed
Private proxy	✘	✓	✘	✓	✓	✓
Unidirectional	✘	✘	✓	✓	✘	✓
Non-interactive	✘	✘	✓	✓	✘	✘
Multi-use	✓	✓	✘	✘	✓	✘
Transparent	✘	✓	✘[1]	✘	✓	✘
Non-transitive	✘	✘	✓	✓	✘	✓
Temporary	✘	✘	✘	✓[2]	✘	✓
Standard model proof	✘	✘	✘	✘	✓[3]	✓

(a) Our design is different from the previous random-oracle based scheme in [1] which maps signatures from $\mathbb{Z}_p$ to $\mathbb{G}$ equipped with bilinear maps and inherently requires non-interactive proof-of-knowledge.
(b) Our notion of compartment signatures may find applications other than proxy re-signatures.

4. We instantiate our generic construction and prove its security in the standard model. This is the first known unidirectional proxy re-signature scheme secure in the standard model. The only other scheme without random oracles was proposed by Shao *et al.* [10] but it is insecure (see Section 4) and is furthermore not unidirectional: it is understood [1] that unidirectional schemes are more challenging to design than bidirectional ones.
5. We discuss a new approach for revoking delegations without changing the global parameter.

Table 1 compares our proxy re-signature scheme with previous work.

2 Definitions

2.1 Bilinear Pairings and Existentially Unforgeable Signatures

Let $\mathbb{G}$ and $\mathbb{G}_T$ be two (multiplicative) cyclic groups of prime order p. Let g be a generator of $\mathbb{G}$. A bilinear map $\hat{e}(\cdot,\cdot) : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ has the following properties:

1. *Bilinearity*: For all $u, v \in \mathbb{G}$ and for all $a, b \in \mathbb{Z}_p$, $\hat{e}(u^a, v^b) = \hat{e}(u, v)^{ab}$.
2. *Non-degeneracy*: $\hat{e}(g, g) \neq 1$.
3. *Computability*: It is efficient to compute $\hat{e}(u, v)$ for all $u, v \in \mathbb{G}$.

A standard signature scheme is a triplet of polynomial time algorithms:

- KeyGen: On input the security parameter 1^k, it outputs a key pair pk/sk.
- Sign: On input a private key sk, and a message m, it outputs a signature σ.
- Verify: On input a public key pk, a message m and a signature σ, it outputs 1 if and only if σ is a valid signature on m under public key pk, 0 otherwise.

Definition 1 (Existential Unforgeability for Signatures [8]). *A signature scheme is existentially unforgeable under adaptive chosen-message attacks if no probabilistic polynomial time (PPT) adversary with access to a signing oracle* $\mathcal{O}_{\mathsf{Sign}}$ *can output with non-negligible probability a pair* (m, σ) *where* m *is not among the queries to the signing oracle, and for which* $\mathsf{Verify}(pk, m, \sigma) = 1$.

[1] Although both transformable and transformed signature are elements in $\mathbb{G} \times \mathbb{Z}_p$, it is easy to distinguish them given the public key and the message.
[2] To revoke the delegated keys, a trusted party is required to broadcast a new global parameter such that all systems user will get an authenticated copy of it.
[3] S_{mb} should be fixed according to what is suggested in this paper.

2.2 Proxy Re-signatures

A proxy re-signature scheme is a quintuple of polynomial time algorithms:

- KeyGen, Sign, and Verify form the key generation, signing and verification algorithms of a standard signature scheme.
- ReKey: On input of (an optional) delegatee's private key sk_A, a delegator's private key sk_B, and the corresponding public keys (pk_A, pk_B), it outputs a transformation key $rk_{A\to B}$ for the proxy to transform A's signature into B's signatures. ReKey may be either deterministic or probabilistic. If the input sk_A is mandatory, the scheme is *interactive*.
- ReSign: On input of a transformation key $rk_{A\to B}$, a *transformable* signature σ, a public key pk_A and a message m, it outputs a signature signed by B if σ is a valid signature signed by A. It may be deterministic or probabilistic.

Some properties of the proxy re-signature schemes [1], that may or may not captured by the above defined framework, are elaborated as follows.

1. **Private Proxy:** The transformation key may be inherently *public* (for example, seeing an original signature signed by B and a transformed signature signed by A give the knowledge of $rk_{B\to A}$), We remark that for the case of *private* transformation key, the key may have public term as well, which can be viewed as a re-signature public/private key pair.
2. **Non-interactive:** If the transformation key can be created without A's help, i.e. ReKey does not require the input of sk_A, the scheme is *non-interactive*.
3. **Unidirectional:** The transformation key may be *unidirectional* or *bidirectional*. In the later case, the knowledge of $rk_{A\to B}$ alone allows the efficient computation of $rk_{B\to A}$. A bidirectional scheme must be interactive since both parties' private key are necessary for the two-way delegation.
4. **Multi-use:** For ReSign algorithm, if the scheme only supports *single-use*, the input signature is required to be an untransformed one outputted by Sign but not by ReSign. For *multi-use* scheme, a signature outputted by ReSign can be further transformed by applying ReSign again.
5. **Transparent:** Transparency means the signatures generated by Sign and the signatures generated by ReSign are computationally indistinguishable.
6. **Non-transitive:** The proxy alone cannot re-delegate signing rights. For example, knowing both $rk_{C\to B}$ and $rk_{B\to A}$ cannot help producing $rk_{C\to A}$.
7. **Temporary:** If one do not want to place trust on the proxy that it will perform the transformation according to the instruction, a *temporary* transformation key can be made expired without changing the public key.

2.3 Review of the Security Model for Proxy Re-signatures

We consider a normal sense of existential unforgeability [8], such that a new signature for a previously signed message is not considered as a valid forgery.

The security model formalized by Ateniese and Hohenberger [1] considers both external and internal security. External security models attack launched

from parties outside the system (i.e. neither the delegation partners nor the proxy), while internal security models those from parties inside the system, such as the proxy, another delegation partner, or a collusion between them.

Suppose there are $n+1$ users with key pairs (pk_i, sk_i) for $i \in \{0, 1, \cdots, n\}$, without loss of generality we assume the adversary aims to compromise the security of user 0; we first define the oracles that an adversary has access to:

- $\mathcal{O}_{\mathsf{Sign}}\ (j, m)$: takes as input an index $0 \leq j \leq n$ and a message $m \in M$, and returns the output of $\mathsf{Sign}\ (sk_j, m)$.
- $\mathcal{O}_{\mathsf{ReSign}}\ (i, j, m, \sigma)$: takes as input two distinct indices $1 \leq i, j \leq n$, a message m and a signature σ, returns $\mathsf{ReSign}\ (\mathsf{ReKey}(sk_i, sk_j, pk_i, pk_j), \sigma, pk_i, m)$.
- $\mathcal{O}_{\mathsf{ReKey}}\ (i, j)$: takes as input two distinct indices $1 \leq i, j \leq n$, and returns the output of $\mathsf{ReKey}\ (pk_i, sk_i, pk_j, sk_j)$.

External attacks assume accesses to $\mathcal{O}_{\mathsf{Sign}}()$ and $\mathcal{O}_{\mathsf{ReSign}}()$.

Definition 2 (External Security). *A proxy re-signature scheme is secure against external adversaries $\mathcal{A}_{\mathrm{ext}}(k)$ if the probability of getting 1 in the following experiment is a negligible function of k.*

Experiment $\mathbf{Exp}_{\mathcal{A}_{\mathrm{ext}}}(k)$
$\{(pk_i, sk_i) \xleftarrow{\$} \mathsf{KeyGen}(1^k)\}_{i \in \{0,\cdots,n\}}$
$(m, \sigma) \xleftarrow{\$} \mathcal{A}_{\mathrm{ext}}^{\mathcal{O}_{\mathsf{Sign}}(\cdot,\cdot), \mathcal{O}_{\mathsf{ReSign}}(\cdot,\cdot,\cdot,\cdot)}(\{pk_i\}_{i \in \{0,\cdots,n\}})$
If $m \in \mathcal{Q}_{\mathrm{ext}}$ *then return 0*
Return $\mathsf{Verify}(pk_0, m, \sigma)$.

where $\mathcal{Q}_{\mathrm{ext}}$ is defined as the set including any message m corresponding to which $\mathcal{A}_{\mathrm{ext}}$ has queried $\mathcal{O}_{\mathsf{Sign}}(0, m)$ or $\mathcal{O}_{\mathsf{ReSign}}(\cdot, 0, m, \cdot)$.

Internal attacks allow accesses to $\mathcal{O}_{\mathsf{Sign}}()$ and $\mathcal{O}_{\mathsf{ReKey}}()$. Within the model of internal security, Ateniese and Hohenberger [1] gave three categorizations:

Definition 3 (Limited-Proxy Security). *A proxy re-signature scheme is secure against limited-proxy adversaries $\mathcal{A}_{\mathrm{pxy}}(k)$ if the probability of getting 1 in the following experiment is a negligible function of k.*

Experiment $\mathbf{Exp}_{\mathcal{A}_{\mathrm{pxy}}}(k)$
$\{(pk_i, sk_i) \xleftarrow{\$} \mathsf{KeyGen}(1^k)\}_{i \in \{0,\cdots,n\}}$
$(m, \sigma) \xleftarrow{\$} \mathcal{A}_{\mathrm{pxy}}^{\mathcal{O}_{\mathsf{Sign}}(\cdot,\cdot), \mathcal{O}_{\mathsf{ReKey}}(\cdot,\cdot)}(\{pk_i\}_{i \in \{0,\cdots,n\}})$
If $m \in \mathcal{Q}_{\mathrm{pxy}}$ *then return 0*
Return $\mathsf{Verify}(pk_0, m, \sigma)$.

where $\mathcal{Q}_{\mathrm{pxy}}$ is the set including any message m corresponding to which $\mathcal{A}_{\mathrm{pxy}}$ has queried $\mathcal{O}_{\mathsf{Sign}}(\star, m)$, for $\star = 0$ or any $\star$ where $\mathcal{O}_{\mathsf{ReKey}}(\star, 0)$ has been queried.

Definition 4 (Delegatee Security). *A proxy re-signature scheme is delegatee-secure against collusion of delegator and proxy adversaries $\mathcal{A}_{\mathrm{dte}}(k)$ if the probability of getting 1 in the following experiment is a negligible function of k.*

Experiment $\mathbf{Exp}_{\mathcal{A}_{\mathrm{dte}}}(k)$
$\{(pk_i, sk_i) \xleftarrow{\$} \mathsf{KeyGen}(1^k)\}_{i\in\{0,\cdots,n\}}$
$(m,\sigma) \xleftarrow{\$} \mathcal{A}_{\mathrm{dte}}^{\mathcal{O}_{\mathsf{Sign}}(0,\cdot),\mathcal{O}_{\mathsf{ReKey}}(\cdot,t)}(pk_0, \{pk_i, sk_i\}_{i\in\{1,\cdots,n\}})$
If $m \in \mathcal{Q}_{\mathrm{dte}}$ *then return* 0
Return $\mathsf{Verify}(pk_0, m, \sigma)$.

where $t \neq 0$ *and* $\mathcal{Q}_{\mathrm{dte}}$ *is defined as the set including any message* m *corresponding to which* $\mathcal{A}_{\mathrm{dte}}$ *has queried* $\mathcal{O}_{\mathsf{Sign}}(0, m)$.

Definition 5 (Delegator Security). *A proxy re-signature scheme is delegator-secure against collusion of delegatee and proxy adversaries* $\mathcal{A}_{\mathrm{dtr}}(k)$ *if the probability of getting* 1 *in the following experiment is a negligible function of* k.

Experiment $\mathbf{Exp}_{\mathcal{A}_{\mathrm{dtr}}}(k)$
$\{(pk_i, sk_i) \xleftarrow{\$} \mathsf{KeyGen}(1^k)\}_{i\in\{0,\cdots,n\}}$
$(m,\sigma) \xleftarrow{\$} \mathcal{A}_{\mathrm{dtr}}^{\mathcal{O}_{\mathsf{Sign}}(0,\cdot),\mathcal{O}_{\mathsf{ReKey}}(\cdot,\cdot)}(pk_0, \{pk_i, sk_i\}_{i\in\{1,\cdots,n\}})$
If $m \in \mathcal{Q}_{\mathrm{dtr}}$ *then return* 0
Return $\mathsf{Verify}(pk_0, m, \sigma)$.

where σ *is a* "untransformed" signature *and* $\mathcal{Q}_{\mathrm{dtr}}$ *is defined as the set including any message* m *corresponding to which* $\mathcal{A}_{\mathrm{dtr}}$ *has queried* $\mathcal{O}_{\mathsf{Sign}}(0, m)$.

An untransformed signature is one which is outputted by the Sign algorithm but not ReSign. Delegator security will be described in details in the next section.

3 Models for Different Proxy Re-signature Schemes

3.1 Public Versus Private Proxy for External Security

One may think the notion of external security is weaker than the internal security, since the oracle $\mathcal{O}_{\mathsf{ReKey}}()$ is more powerful than $\mathcal{O}_{\mathsf{ReSign}}()$. Nevertheless, these two notions are actually for two different kinds of proxy re-signature schemes.

The former notion suggests a justification whether a proxy re-signature scheme is a private or a public one, i.e. whether the transformation key is inherently public knowledge. Suppose there is a scheme which is easy to recover the transformation key $rk_{B\to A}$ from σ and σ' where σ is a signature on m under the public key pk_B and σ' is a re-signature transformed from pk_B to pk_A, (e.g. the BBS scheme [2], and S_{uni} in [1] possess this property). Consider the below attack:

1. Query $\mathcal{O}_{\mathsf{Sign}}(B, m)$ to get σ, where $B \in \{0, n\}$;
2. Query $\mathcal{O}_{\mathsf{ReSign}}(B, A, m, \sigma)$ to get σ', where $B \in \{0, n\}\backslash\{A\}$;
3. Recover $rk_{B\to A}$ from σ and σ';
4. Query $\mathcal{O}_{\mathsf{Sign}}(B, m^*)$ to get σ^*;
5. Return $\sigma^* = \mathsf{ReSign}(rk_{B\to A}, \sigma^*, pk_B, m^*)$ as a forged signature of A on m^*.

Since the adversary made no query of $\mathcal{O}_{\mathsf{Sign}}(A, m^*)$ and $\mathcal{O}_{\mathsf{ReSign}}(\star, A, m^*, \sigma^*)$ for signature σ^*, it is a valid forgery under the external security definition. Note that for a public-proxy scheme, $\mathcal{O}_{\mathsf{Sign}}(\star, m)$ and $\mathcal{O}_{\mathsf{ReSign}}(\star, 0, m, \sigma)$ together return essentially the answer of $\mathcal{O}_{\mathsf{ReKey}}(\star, 0)$. In the limited-proxy definition, the adversary is not allowed to issue $\mathcal{O}_{\mathsf{Sign}}(\star, m)$ query if $\mathcal{O}_{\mathsf{ReKey}}(\star, 0)$ is made. It is exactly the difference between the notions of internal security and limited-proxy.

To conclude, for a public-proxy scheme, one should consider only internal security but not external security. This point is also mentioned in [1]. On the other hand, one must take both external security and internal security into account for a private-proxy scheme. In other words, the property of private-proxy is modeled by the notion of external security.

3.2 Limited-Proxy Security Versus Delegatee Security

One may treat the adversary $\mathcal{A}_{\mathrm{dte}}$ in the delegatee security model is more powerful than the adversary $\mathcal{A}_{\mathrm{pxy}}$ in the limited-proxy security model, due to the fact that the former is equipped with n private signing keys but the latter is only provided with signing oracle accesses. We stress that it is not the case since the allowed queries for a valid forgery are different. In the limited-proxy model, if $\mathcal{A}_{\mathrm{pxy}}$ has not queried $\mathcal{O}_{\mathsf{Sign}}(\star, m)$, it is entitled to query $\mathcal{O}_{\mathsf{ReKey}}(\star, 0)$. However, in the delegatee model, $\mathcal{A}_{\mathrm{dte}}$ cannot ask for $\mathcal{O}_{\mathsf{ReKey}}(\star, 0)$ at all. The limited-proxy model and the delegatee security model are actually orthogonal to each other.

3.3 Delegatee Security for Non-interactive Schemes

Recall that a scheme's non-interactivity is determined by whether the delegatee's private key is required in constructing the transformation key, thus a scheme's non-interactivity affects the delegatee security notion of Definition 4. In more detail, in a non-interactive scheme, the transformation-key can be computed without the help of the delegatee, hence transformation-key extraction queries $\mathcal{O}_{\mathsf{ReKey}}()$ give the adversary nothing more about the delegatee than what the adversary can compute by himself, so this oracle can be safely removed.

3.4 Security for Transparent and Non-transparent Schemes

The unforgeability of a proxy re-signatures scheme also depends on whether the scheme is transparent, i.e. whether the signature generated by the usual signing algorithm (termed as the "untransformed" signature) and that by the re-signing algorithm (termed as the "transformed" signature) are indistinguishable.

In the definition of delegator security [1], only a forgery of the "first-level" (i.e. untransformed) signature is considered as a valid forgery. From a first glance, it seems that the notion is tightly coupled with a non-transparent scheme.

Another point is about the possible kind of forgery. For *transparent* schemes, the untransformed signature and the transformed signature are computationally indistinguishable, and hence there is essentially only a single kind of forgery.

However, for non-transparent scheme, we should make a fine distinction between forgery for these two types of distinguishable signatures; and it seems that this aspect was ignored in the Ateniese-Hohenberger definition of delegator security. In the following, we will show exactly what security properties one should expect for different scheme and different type of forgery.

Transparent Scheme. The case for a transparent scheme is simpler since there is no distinction between the forgery of an untransformed signature or a transformed signature. Suppose the private keys of all other systems users are known to the adversary (i.e., $\{sk_i : i \in \{1, \cdots n\}\}$ are given), allowing the adversary to query for $\mathcal{O}_{\mathsf{ReKey}}(\star, 0)$ corresponds to a trivial forgery. Hence such queries should not be allowed, and it is exactly the definition of delegatee security.

If the untransformed signature and the transformed signature are indistinguishable (i.e. "forgery of an untransformed signature" in the definition just means "forgery of a signature"), the only difference between the definitions of delegatee and delegator security is that no $\mathcal{O}_{\mathsf{ReKey}}(\star, 0)$ is allowed in the former. With all other users' private keys, asking $\mathcal{O}_{\mathsf{ReKey}}(\star, 0)$ means a trivial forgery.

For an attack launched by a proxy who are not colluding with any users, a signature σ on the message m under the public key of pk_0, should considered be a non-trivial forgery if the adversary never ask for $\mathcal{O}_{\mathsf{Sign}}(0, m)$, and not both of $\mathcal{O}_{\mathsf{ReKey}}(\star, 0)$ and $\mathcal{O}_{\mathsf{Sign}}(\star, m)$ queries. Disallowing the $\mathcal{O}_{\mathsf{ReKey}}(\star, 0)$ query is covered in the definition of the delegatee security (where the adversary is given the signers' private keys to realize the signing oracles), while the definition of limited-proxy security models exactly the scenario that no $\mathcal{O}_{\mathsf{Sign}}(\star, m)$ queries are allowed. The below definitions conclude our discussion on transparent scheme.

Definition 6 (Transparent Public). *A transparent public-proxy re-signature scheme is secure if it is limited-proxy-secure and delegatee-secure.*

Definition 7 (Transparent Private). *A transparent private-proxy re-signature scheme is secure if it is external-secure, limited-proxy-secure and delegatee-secure.*

Non-Transparent Scheme. In order to distinguish between an untransformed signature and a transformed one, we suppose a description of an algorithm $\mathsf{isTransformed}()$ is (implicitly) included in the system parameter, which answers the predicate if a signature is generated from the ReSign algorithm.

Firstly, for a specific proxy re-signature scheme, an untransformed signature under a public key on a message may be publicly transformable to a transformed signature under the same public key on the same message, so we consider a signature on message m is a trivial forgery if $\mathcal{O}_{\mathsf{Sign}}(0, m)$ has been asked.

For the forgery of a transformed signature, trivial forgery includes issuing $\mathcal{O}_{\mathsf{ReKey}}(\star, 0)$ queries when all other users' private keys are known; or in the case that no private keys are leaked, not both of $\mathcal{O}_{\mathsf{ReKey}}(\star, 0)$ and $\mathcal{O}_{\mathsf{Sign}}(\star, m)$ queries are made. These just correspond to delegatee security and limited-proxy security.

For an untransformed signature forgery, since the $\mathcal{O}_{\mathsf{ReKey}}$ are supposed to be related to transformed signatures only, answering any such queries should

be "safe", and this is exactly the definition of delegator security proposed in [1]. In other words, even though the definition is coupled with a non-transparent scheme, Ateniese and Hohenberger indeed provide a "right" definition of security.

Definition 8 (Non-Transparent Public-Proxy). *A non-transparent public-proxy re-signature scheme is secure if it is limited-proxy-secure, delegatee-secure and delegator-secure.*

Definition 9 (Non-Transparent Private-Proxy). *A non-transparent private-proxy re-signature scheme is secure if it is external-secure, limited-proxy-secure, delegatee-secure and delegator-secure.*

3.5 Security for Bidirectional Schemes

The notion of limited-proxy security is refined for bidirectional schemes [1].

Definition 10 (Bidirectional Limited-Proxy Security). *A bidirectional proxy re-signature scheme is secure against bidirectional limited-proxy adversaries if the probability of getting* 1 *in the below experiment is negligible in* k*.*

Experiment $\mathbf{Exp}_{\mathcal{A}^{\mathrm{bi}}_{\mathrm{pxy}}}(k)$
$\{(pk_i, sk_i) \xleftarrow{\$} \mathsf{KeyGen}(1^k)\}_{i\in\{0,\cdots,n\}}$
$(m, \sigma) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_{\mathsf{Sign}}(\cdot,\cdot),\mathcal{O}_{\mathsf{ReKey}}(\cdot,\cdot)}_{\mathrm{pxy}}(\{pk_i\}_{i\in\{0,\cdots,n\}})$
If $m \in \mathcal{Q}^{\mathrm{bi}}_{\mathrm{pxy}}$ *then return 0*
Return $\mathsf{Verify}(pk_0, m, \sigma)$.

where $\mathcal{Q}^{\mathrm{bi}}_{\mathrm{pxy}}$ *is defined as the set including any message* m *which* $\mathcal{A}_{\mathrm{pxy}}$ *has queried* $\mathcal{O}_{\mathsf{Sign}}(\star, m)$*, for any* $\star \in \{0, \cdots, n\}$*.*

The reason for prohibiting all signing queries on the forgery message m is that the adversary is free to ask for any transformation key, which gives the power to transform any signature of one to a signature of another.

In a bidirectional scheme, once a delegation is made, both involved parties mutually delegate to each other, so the notion of delegatee security does not apply since one cannot play only the role "delegatee" without being a delegator.

For delegator security, it is suggested in [1] that "this property is not required" and "it does not seem likely to be achievable". The reason behind such claims may be based on the fact that the all bidirectional schemes considered in [1] (specifically, the BBS scheme [2], and S_{bi} in [1]) are transparent.

We argue that the above claims only make sense for transparent schemes. For a non-transparent scheme, even though the transformation keys are bidirectional, it is reasonable to expect these keys should not enable an adversary to come up with a forgery of a transformable (i.e. untransformed) signature. In view of this, we give the following definitions.

Definition 11. *A bidirecitonal transparent public-proxy re-signature scheme is secure if it is bidirecitonal limited-proxy-secure.*

Definition 12. *A bidirecitonal transparent private-proxy re-signature scheme is secure if it is external-secure and bidirecitonal limited-proxy-secure.*

Definition 13. *A bidirectional non-transparent public-proxy re-signature scheme is secure if it is bidirectional limited-proxy-secure, and delegator-secure.*

Definition 14. *A bidirectional non-transparent private-proxy re-signature scheme is secure if it is external-secure, bidirectional limited-proxy-secure, and delegator-secure.*

3.6 Transitivity

Consider the below attack on limited-proxy security for a transitive scheme:

1. Query $\mathcal{O}_{\mathsf{Sign}}(C, m)$ to get σ, where $C \in \{0, n\}$;
2. Query $\mathcal{O}_{\mathsf{ReKey}}(C, B)$ to get $rk_{C\to B}$, where $B \in \{0, n\}\backslash\{C\}$;
3. Query $\mathcal{O}_{\mathsf{ReKey}}(B, A)$ to get $rk_{B\to A}$, where $A \in \{0, n\}\backslash\{B, C\}$;
4. Compute $rk_{C\to A}$ from $rk_{C\to B}$ and $rk_{B\to A}$ by the transitivity;
5. Return $\sigma^* = \mathsf{ReSign}(rk_{C\to A}, \sigma, pk_C, m)$ as a forged signature of A on m.

Since the adversary made no query of $\mathcal{O}_{\mathsf{Sign}}(A, m)$ and no $\mathcal{O}_{\mathsf{ReKey}}(C, A)$ query when $\mathcal{O}_{\mathsf{Sign}}(C, m)$ has been made. It is a valid forgery under the limited-proxy security definition. To conclude, limited-proxy security implies non-transitivity. The definition of limited-proxy security should be extended for transitive schemes. Informally, it adds more restrictions in $\mathcal{O}_{\mathsf{ReKey}}$ queries to avoid trivial forgery.

4 Analysis of a Bidirectional Proxy Re-signature Scheme

4.1 Review

We review the public-key-based bidirectional proxy re-signature scheme S_{mb} proposed by Shao *et al.* [10], which security was shown in the standard model.

The global parameters are $\langle \mathbb{G}, \mathbb{G}_T, \hat{e}(\cdot,\cdot), p, \eta, g, u', \mathbf{U}, H_u(\cdot)\rangle$ where $\mathbb{G}, \mathbb{G}_T$, $\hat{e}(\cdot,\cdot), p$ are defined as in Section 2, g, u' are two generators of $\mathbb{G}$, $\mathbf{U} = (u_1, \cdots, u_\eta)$ is a vector with η random elements from $\mathbb{G}$, $H_u : \mathbb{G}_T \to \{0,1\}^\eta$ is a collision-resistant hash function. Let $bit(i, \mathfrak{s})$ be the i-th bit of a bit-string $\mathfrak{s}$.

- $\mathsf{KeyGen}(1^k)$: Pick a $x \in \mathbb{Z}_p$, the public/private key pair is given by $(\hat{e}(g, g^x), g^x)$.
- $\mathsf{ReKey}(sk_A, sk_B)$: On input of two private keys, output $rk_{A\to B} = sk_B/sk_A$.
- $\mathsf{Sign}(sk, m)$: On input a secret key $sk = g^\alpha$ and a message m, pick a random $r_m \in \mathbb{Z}_p$ and output $\langle \mathfrak{A} = g^\alpha (u' \prod u_i^{bit(i,H_u(m))})^{r_m}$ and $\mathfrak{B} = g^{r_m}\rangle$.
- $\mathsf{ReSign}(rk_{A\to B}, \sigma, pk_A, m)$: On input of a transformation key $rk_{A\to B} = \beta/\alpha$, a signature $\sigma = \langle \mathfrak{A}, \mathfrak{B}\rangle$, the original signer's public key pk_A and a message m, first verify if σ is valid by $\mathsf{Verify}(pk_A, m, \sigma)$. If it does not verify, return $\perp$, otherwise output $\langle \mathfrak{A}^{rk_{A\to B}}, \mathfrak{B}^{rk_{A\to B}}\rangle = \langle g^\beta (u' \prod u_i^{bit(i,H_u(m))})^{r_m\beta/\alpha}, g^{r_m\beta/\alpha}\rangle$.
- $\mathsf{Verify}(pk, m, \sigma)$: On input of a public key pk, a message m, and a signature σ, parse σ as $\langle \mathfrak{A}, \mathfrak{B}\rangle \in \mathbb{G}^2$, output 1 if $\hat{e}(\mathfrak{A}, g) = pk \cdot \hat{e}(u' \prod u_i^{bit(i,H_u(m))}, \mathfrak{B})$.

4.2 Attacks

A formal description of the oracles $\mathcal{O}_{UKeyGen}$, $\mathcal{O}_{CKeyGen}$ and $\mathcal{O}_{ReSign}$ considered by the Shao *et al.* model can be found in [10]. However, their meaning should be clear enough from the context. Our first attack requires finding four hash values h_1, h_2, h_3, h_4 such that $h_4 = h_1 + h_2 - h_3$ but $h_1 \neq h_4$. For η-bit message, it can be shown that the probability of finding these messages is $\eta(2/16)(6/16)^{\eta-1}$. To see, we have $0+0-0, 0+1-0, 0+1-1, 1+0-0, 1+0-1, 1+1-1 \in \{0,1\}$ (as a counter example, $0+0-1 \notin \{0,1\}$); and two possibilities give a different bit after addition and subtraction, specifically, $0+1-0 = 1 \neq 0$ and $1+0-1 = 0 \neq 1$. We suppose $\mathfrak{m}_1, \mathfrak{m}_2, \mathfrak{m}_3, \mathfrak{m}_4$ are the messages such that $H_u(\mathfrak{m}_i) = h_i, i \in \{1, \cdots 4\}$.

1. Get the public keys.
   - Call $\mathcal{O}_{UKeyGen}$ to get an uncorrupted key pk_0, this is our target of attack.
   - Call $\mathcal{O}_{CKeyGen}$ to get a corrupted public/private key pair (pk_1, sk_1).
2. Sign the first message and ask the oracle to re-sign it.
   - Define $U_1 = u' \prod {u_i}^{bit(i,\mathfrak{m}_1)}$, which is to be signed by sk.
   - Pick $r \in_R \mathbb{Z}_p$, issue $(pk_1, pk_0, \mathfrak{m}_1, (g^{sk_1} \cdot U_1^r, g^r))$ to $\mathcal{O}_{ReSign}$ to get $\mathsf{ReSign}(\mathsf{ReKey}(sk_1, sk_0), \sigma, pk_1, \mathfrak{m}_1) = (\mathfrak{A}, \mathfrak{B}) = (g^{sk_0} \cdot U_1^{r(sk_0/sk_1)}, g^{r(sk_0/sk_1)})$.
3. Sign the second message and ask the oracle to re-sign it.
   - Let $U_2 = u' \prod {u_i}^{bit(i,\mathfrak{m}_2)}$, submit $(pk_1, pk_0, \mathfrak{m}_2, (g^{sk_1} \cdot U_2^r, g^r))$ to $\mathcal{O}_{ReSign}$ to get $(\mathfrak{A}', \mathfrak{B}') = (g^{sk_0} \cdot U_2^{r(sk_0/sk_1)}, g^{r(sk_0/sk_1)})$.
4. Sign the third message and ask the oracle to re-sign it.
   - Let $U_3 = u' \prod {u_i}^{bit(i,\mathfrak{m}_3)}$, submit $(pk_1, pk_0, \mathfrak{m}_3, (g^{sk_1} \cdot U_3^r, g^r))$ to $\mathcal{O}_{ReSign}$ to get $(\mathfrak{A}'', \mathfrak{B}'') = (g^{sk_0} \cdot U_3^{r(sk_0/sk_1)}, g^{r(sk_0/sk_1)})$.
5. Output the forgery $\sigma^* = (\mathfrak{A} \cdot \mathfrak{A}'/\mathfrak{A}'', \mathfrak{B}) = (g^{sk_0} \cdot (U^*)^{r(sk_0/sk_1)}, g^{r(sk_0/sk_1)})$ where $U^* = U_1 \cdot U_2/U_3 = u' \prod {u_i}^{b_i}$, $b_i = bit(i, \mathfrak{m}_1) + bit(i, \mathfrak{m}_2) - bit(i, \mathfrak{m}_3)$.

By our choices of messages, we have $\forall i \in \{1, \cdots, \eta\}, b_i \in \{0,1\}$ and $\exists j \in \{1, \cdots, \eta\}, b_j \neq bit(j, \mathfrak{m}_1)$. It is easy to see that σ^* is a valid signature on $\mathfrak{m}_4$ under an uncompromised key where the message being signed did not appear in any $\mathcal{O}_{ReSign}$ query (and any $\mathcal{O}_{Sign}$ query, which does not appear at all).

The above attack manipulates the bit strings such that an addition and then a subtraction gives a valid bit string different from the original one. Taking a step further, we can cancel the whole message part and get back the private key!

1. Get an uncorrupted key pk_0, and a corrupted public/private key pair (pk_1, sk_1).
2. Sign an arbitrary message $\mathfrak{m}$ and ask the oracle to re-sign it.
   - Pick $r \in_R \mathbb{Z}_p$, issue $(pk_1, pk_0, \mathfrak{m}, (g^{sk_1} \cdot U^r, g^r))$ to $\mathcal{O}_{ReSign}$ to get $(\mathfrak{A}, \mathfrak{B})$, where $\mathfrak{A} = g^{sk_0} \cdot U^{r(sk_0/sk_1)}$, $U = u' \prod {u_i}^{bit(i,\mathfrak{m})}$ ($\mathfrak{B}$ is irrelevant here).
3. Re-randomize the same signature and ask the oracle to re-sign it.
   - Issue $(pk_1, pk_0, \mathfrak{m}, (g^{sk_1} \cdot U^r \cdot U^r, g^r \cdot g^r))$ to $\mathcal{O}_{ReSign}$ to get $(\mathfrak{A}', \mathfrak{B}')$, where $\mathfrak{A}' = g^{sk_0} \cdot U^{2r(sk_0/sk_1)}$.
4. Recover the secret key by $\mathfrak{A} \cdot \mathfrak{A}/\mathfrak{A}' = g^{sk_0}$.

One can prevent this attack by requiring the proxy to keep track of all the messages being processed and does not re-sign if the same message appears, but it is not practical and not the best we can hope for.

The previous proposals of bidirectional proxy re-signature (which include BBS [2] and S_{bi} [1]) all use a deterministic ReSign algorithm. The scheme we attack (S_{mb} in [10]) also has a deterministic ReSign, possibly because recovering the transformation key from the transformed signature needs computing discrete logarithm, since ReSign exponentiates the signature with the transformation key.

The source of the insecurity of S_{mb} is due to the fact that the "random" factor in the signature outputted by the transformation is uniquely determined by the keys involved in the transformation, while the security of the scheme requires the random factors in any signature to be uniformly distributed. Specifically, their simulation of $\mathcal{O}_{ReSign}$ just returns the answer of (probabilistic) $\mathcal{O}_{Sign}$ if only one of the concerned keys are corrupted (the scenario for our attacks), which is different from what their (deterministic) ReSign algorithm is doing. Thus, a possible fix is to re-randomize the signature outputted by the ReSign algorithm.

5 Homomorphic Compartment Signatures

Before delving into details of our constructions of proxy re-signature schemes, we review two existing concepts – homomorphic signatures and hierarchical signatures, and generalize these to the notion of homomorphic compartment signatures, which may be of independent interest.

5.1 Hierarchical Signatures and Homomorphic Signatures

Hierarchical Signatures. In an ℓ-level hierarchical signature [7], the message being signed is actually an ℓ blocks message. The crucial property is that a signature on $(m_1, m_2, \cdots, m_i)$ can act as a restricted private key that enables the signing of any extension, $(m_1, m_2, \cdots, m_i, m_j)$, of which the original one is a prefix. Of course we require $1 \leq i \leq j \leq \ell$. As noted in [5], an ℓ-level hierarchical signature is actually equivalent to an $(\ell-1)$-hierarchical identity-based signature (HIBS) [7], in which the first $\ell - 1$ levels are viewed as the components of a hierarchical identity, and the last level is for the message being signed. Our construction of proxy re-signature in Section 6 uses a two-level hierarchy.

Homomorphic Signatures. In our definition of homomorphic property for probabilistic signature schemes, the homomorphism only holds for the randomness and the secret key being used, but not the messages (in contrast to [9]).

Definition 15 (Homomorphic Signature). *A signature scheme is homomorphic if* $\forall SK, SK' \in \mathcal{K}; \forall\ m \in \mathcal{M}; \forall\ r, r' \in \mathcal{R}$, *such that we have* $\mathsf{HSign}_{SK}(m; r) \cdot \mathsf{HSign}_{SK'}(m; r') = \mathsf{HSign}_{SK+SK'}(m; r + r')$ *where the signature space* $\mathcal{S}$ *is a group represented multiplicatively, and the rest of them (the key space* $\mathcal{K}$*, the message space* $\mathcal{M}$*, and the random coin space* $\mathcal{R}$*) are additive groups.*

Note that the random factor r is not the input of HSign, but is chosen by HSign itself. Homomorphic property over the private keys has also been exploited in other settings, such as untrusted update in forward-secure signature [4].

5.2 Generalizing Hierarchical Signatures

In our approach of constructing proxy re-signature schemes, we need something more general than a hierarchical signature, which we term as a *compartment signature*. In an ℓ-block compartment signature, similar to an ℓ-level hierarchical signature, ℓ message blocks can be signed. The difference from hierarchical signature is that we can add message blocks in an order independent of their positions, instead of following the sequential order of their corresponding positions. Specifically, a signature on $m_{i_1}, m_{i_2}, \cdots, m_{i_n}$ where $1 \leq i_1 < i_2 < \cdots < i_n \leq \ell$ can act as a restricted private key that enables the signing of any insertion of m_j to the original one, i.e. $j \in (\{1, \cdots, \ell\} \backslash \{i_1, \cdots, i_n\})$. Consider $\ell = 2$, and let $\star$ be a symbol denoting a "null" message, that means the secret key can sign on either $(m_1, \star)$, $(\star, m_2)$ or (m_1, m_2), while both the signature on $(m_1, \star)$ and the signature on $(\star, m_2)$ can act as a restricted private key to sign on (m_1, m_2). We remark that the messages can be completely arbitrary, e.g. $m_1 \neq m_2$.

Now we discuss from a high level point of view the idea of using signature as a signing key. Specifically, the function HSign can be viewed as a procedure to sign on an ℓ-block message m in one shot. The compartment-signing property means there exists a polynomial-time algorithm that takes a signature of i message blocks to give a signature on j message blocks where $i < j$ and the former set of message blocks is a subset of the latter. Suppose the factor r_i is the randomness involved in signing m_i $(\forall i \in \{1, \cdots, \ell\})$. One can get $\sigma = \mathsf{HSign}_{SK}(m_1, \cdots, m_\ell; r_1, \cdots, r_\ell)$ in various ways. For example, one can first compute $\sigma' = \mathsf{HSign}_{SK}(\star, m_2, \star, \cdots; 0, r_2, 0, \cdots)$, and use σ' (without the help of SK) to compute $\mathsf{HSign}_{SK}(\star, m_2, \star, \cdots, m_\ell; 0, r_2, 0, \cdots, r_\ell)$ and eventually obtain a "full" signature $\sigma = \mathsf{HSign}_{SK}(m_1, m_2, m_3, \cdots, m_\ell; r_1, r_2, r_3, \cdots, r_\ell)$.

This notion can be further enriched by the homomorphic property.

Definition 16 (Homomorphic Compartment Signature). *An ℓ-block compartment signature scheme is homomorphic if $\forall SK, SK' \in \mathcal{K}$; $\forall m_i \in \mathcal{M}, i \in \{1, \cdots, \ell\}$; and $\forall r_i, r'_i \in \mathcal{R}, i \in \{1, \cdots, \ell\}$, such that*

$$\begin{aligned} &\mathsf{HSign}_{SK}(m_1, \cdots, m_\ell; r_1, r_2, \cdots, r_\ell) \cdot \mathsf{HSign}_{SK'}(m_1, \cdots, m_\ell; r'_1, r'_2, \cdots, r'_\ell) \\ &= \mathsf{HSign}_{SK+SK'}(m_1, \cdots, m_\ell; r_1 + r'_1, r_2 + r'_2, \cdots, r_\ell + r'_\ell) \end{aligned}$$

where the signature space $\mathcal{S}$ is a multiplicative group, and the rest (the key space $\mathcal{K}$, the message space $\mathcal{M}$, and the random coin space $\mathcal{R}$) are additive groups.

5.3 Security for Compartment Signatures

To extend from the unforgeability of a standard signature scheme, we need to first define what is meant by a message *matches with a pattern*.

Definition 17. *A message* $(m_1, m_2, \cdots, m_j)$ *is defined to match with a pattern* $(m'_1, m'_2, \cdots, m'_k)$ *if* $j = k$ *and either* $m_i = m'_i$ *or* $m'_i = \star$, $\forall i \in \{1, \cdots, j\}$.

Definition 18 (Existential Unforgeability for Compartment Signatures). *A compartment signature scheme is existentially unforgeable under adaptive chosen-pattern attacks if no PPT adversary with access to a signing oracle* $\mathcal{O}_{\mathsf{Sign}}$ *can output with non-negligible probability a pair* (m, σ) *where* m *does not match with any of the chosen patterns queried to* $\mathcal{O}_{\mathsf{Sign}}$*, and for which* $\mathsf{Verify}(pk, m, \sigma) = 1$.

5.4 Examples

While compartment signature is more general than hierarchical signature, many hierarchical schemes, such as [3,4,5,7], can be used to give compartment signatures. To see this, these schemes use different parameters for different levels, and uses only commutative multiplications to "glue" up different components. In particular, it is easy to obtain an ℓ-block homomorphic compartment signature scheme from Boyen-Waters ℓ-level hierarchical signature scheme [5]. Our proxy re-signature construction uses 2-block homomorphic compartment signatures. The compartment signature scheme for the 2-block case and its security analysis are deferred to the full version of this paper due to page limitation.

6 Generic Proxy Re-signature Scheme and Instantiation

In this paper, we propose a *unidirectional* proxy re-signature scheme since it is more challenging [1] to design when compared with a bidirectional one and more applicable. As with the previous unidirectional scheme in [1], our scheme supports private proxy and non-transitivity, but our scheme is interactive.

Let $\mathcal{CS}$ be a 2-block compartment signature scheme. Let $H_r(\cdot)$ be a collision-resistant hash function that maps the keyspace $\mathcal{K}$ of $\mathcal{CS}$ to the message space $\mathcal{M}$ of $\mathcal{CS}$. The system parameter also includes all those of $\mathcal{CS}$, in particular, a random coin space $\mathcal{R}$. Let $1_{\mathcal{K}}$ and $0_{\mathcal{R}}$ denote the identity element in $\mathcal{K}$ and $\mathcal{R}$ respectively. Our generic proxy re-signature scheme is defined as follows.

Definition 19 (Generic Proxy Re-Signature Scheme)

- $\mathsf{KeyGen}(1^k)$*: This is the same as the one in* $\mathcal{CS}$.
- $\mathsf{ReKey}(sk_A, sk_B, pk_A, pk_B)$*: With the key pairs of a delegatee* (pk_A, sk_A) *and a delegator* (pk_B, sk_B)*, the transformation key is given by:*
  1. *Compute* $m_{A\to B} = H_r(pk_B)$*;*
  2. *Randomly select* r_A *from* $\mathcal{R}$*, compute* $\sigma^A \leftarrow \mathsf{HSign}_{sk_A}(m_{A\to B}, \star; r_A, 0_{\mathcal{R}})$*;*
  3. *Randomly select* r_B *from* $\mathcal{R}$*, compute* $\sigma^B \leftarrow \mathsf{HSign}_{sk_B}(m_{A\to B}, \star; r_B, 0_{\mathcal{R}})$*;*
  4. *Output transformation key as* $rk_{A\to B} = \sigma^B/\sigma^A$.

  From the homomorphism, $rk_{A\to B} = \mathsf{HSign}_{sk_B - sk_A}(m_{A\to B}, \star; r_B - r_A, 0_{\mathcal{R}})$.
- $\mathsf{Sign}(sk, m)$*: On input a secret key* sk *and a message* m*, randomly pick* r *from* $\mathcal{R}$*, the (transformable) signature is* $\mathsf{HSign}_{sk}(\star, m; 0_{\mathcal{R}}, r)$.

- $\mathsf{ReSign}(rk_{A\to B}, \sigma, pk_A, m)$: *On input of a transformation key* $rk_{A\to B}$*, an untransformed signature* σ*, a public key* pk_A *and a message* m*, first verify if the signature is valid by* $\mathsf{Verify}(pk_A, m, \sigma)$*. If it does not verify, return* $\perp$*, otherwise randomly select* r' *from* $\mathcal{R}$ *and output* $\sigma \cdot rk_{A\to B} \cdot \mathsf{HSign}_{1_{\mathbb{G}_s}}(\star, m; 0_{\mathcal{R}}, r')$.[1]
- $\mathsf{Verify}(pk, m, \sigma)$: *On input of a public key* pk*, a message* m*, and a signature* σ*, use the verification algorithm of* $\mathcal{CS}$ *to verify if* σ *is valid under* pk *on the message* $(\star, m)$ *for transformable signature, or* $(H_r(pk), m)$ *otherwise.*

6.1 Security

Theorem 1. *The (unidirectional, non-transparent and interactive) private-proxy re-signature scheme of Definition 19 is secure, if the underlying 2-block compartment signature* $\mathcal{CS}$ *is existentially unforgeable and* $H_r(\cdot)$ *is collision-resistant.*

The proof can be found in the full version of this paper.

6.2 A Concrete Scheme in the Standard Model

We instantiate our construction with Boyen-Waters hierarchical signature [5].

Most part of the global parameter is the same as reviewed in Section 4.1. We also let $v' \in \mathbb{G}$, $\mathbf{V} = (v_1, \cdots, v_\eta)$ be a vector with η random elements from $\mathbb{G}$, and $H_v : \{0,1\}^* \to \{0,1\}^\eta$ be a collision-resistant hash function.

- $\mathsf{KeyGen}(1^k)$: Pick a $x \in \mathbb{Z}_p$, the public/private key pair is given by $(\hat{e}(g, g^x), g^x)$.
- $\mathsf{ReKey}(sk_A, sk_B, pk_A, pk_B)$: With the key pairs of a delegatee $(pk_A, sk_A = g^\alpha)$ and a delegator $(pk_B, sk_B = g^\beta)$, the transformation key is given by:
  1. Pick a random $r_A \in \mathbb{Z}_p$;
  2. Compute $K_A = g^\alpha (u' \prod u_i^{bit(i, H_u(pk_B))})^{r_A}$ and $R_A = g^{r_A}$;
  3. Pick a random $r_B \in \mathbb{Z}_p$;
  4. Compute $K_B = g^\beta (u' \prod u_i^{bit(i, H_u(pk_B))})^{r_B}$ and $R_B = g^{r_B}$;
  5. Output $rk_{A\to B}$ as $\langle rsk_{A\to B}, rpk_{A\to B}\rangle = \langle K_B/K_A, R_B/R_A\rangle$.
- $\mathsf{Sign}(sk, m)$: On input a secret key $sk = g^\alpha$ and a message m, let $\mathfrak{m}$ be the bit string outputted by $H_v(m)$, the signature is generated as follows.
  1. Pick a random $r_m \in \mathbb{Z}_p$;
  2. Compute $\sigma_0 = g^\alpha (v' \prod v_i^{bit(i, \mathfrak{m})})^{r_m}$ and $\sigma_2 = g^{r_m}$;
  3. Output the signature $\sigma = \langle \sigma_0, \sigma_2 \rangle$.
- $\mathsf{ReSign}(rk_{A\to B}, \sigma, pk_A, m)$: On input of a key $rk_{A\to B} = \langle rsk_{A\to B}, rpk_{A\to B}\rangle$, an untransformed signature $\sigma = \langle \sigma_0, \sigma_2 \rangle$, the original signer's public key pk_A and a message m, first verify if σ is valid by $\mathsf{Verify}(pk_A, m, \sigma)$. If it does not verify, return $\perp$, otherwise compute $\mathfrak{m} = H_v(m)$, pick a random $r' \in \mathbb{Z}_p$ and output $\langle \sigma_0 \cdot rsk_{A\to B} \cdot (v' \prod v_i^{bit(i, \mathfrak{m})})^{r'}, rpk_{A\to B}, \sigma_2 \cdot g^{r'} \rangle$.

[1] The introduction of the term $\mathsf{HSign}_{1_{\mathbb{G}_s}}(\star, m; 0_{\mathcal{R}}, r')$ is for making a public-proxy scheme private. If public-proxy is desired, ReSign just multiplies σ with $rk_{A\to B}$, which should be more efficient than the Sign algorithm of a typical signature scheme.

- $\mathsf{Verify}(pk, m, \sigma)$: On input of a public key pk, a message m, and a purported signature σ, compute $\mathfrak{m} = H_v(m)$, if σ is parsed as $\langle \sigma_0, \sigma_2 \rangle \in \mathbb{G}^2$, output 1 if and only if $\hat{e}(\sigma_0, g) = pk \cdot \hat{e}(v' \prod {v_i}^{bit(i,\mathfrak{m})}, \sigma_2)$; else if σ is parsed as $\langle \sigma_0, \sigma_1, \sigma_2 \rangle \in \mathbb{G}^3$, compute $\mathfrak{u} = H_u(pk)$, output 1 if and only if $\hat{e}(\sigma_0, g) = pk \cdot \hat{e}(u' \prod {u_i}^{bit(i,\mathfrak{u})}, \sigma_1) \cdot \hat{e}(v' \prod {v_i}^{bit(i,\mathfrak{m})}, \sigma_2)$.

We have the following security result, which follows from Theorem 1.

Corollary 1. *The above concrete (unidirectional, non-transparent and interactive) proxy re-signature scheme is secure in the standard model, assuming the intractability of the computational Diffie-Hellman problem and $H_u(\cdot), H_v(\cdot)$ are collision-resistant.*

7 Forward-Security and Temporary Delegation

Forward-secure signature (FSS) schemes modify a secret key over time (while the public key remains the same) such that the exposure of the secret key at a certain time period does not allow forgery of signatures of previous time periods. Using the tree-based construction in [6], it is easy to show that HIBS implies FSS. More precisely, a forward-secure scheme for 2^d time steps can be constructed from a HIBS scheme of depth d. That being said, it is easy to construct a forward-secure proxy re-signature. By using ℓ-block compartment signatures ($\ell > 3$), we can get a FSS scheme with $2^{\ell-3}$ periods, using the first $\ell - 2$ level, while the last 2 levels are used to certify the transformation and the message being signed.

Our construction certifies the delegation relationship by signature, the delegation thus can be made time-limited, i.e. the transformation key created at time t can only enable the transformation of signatures for time t, but not any signature issued at any other time. We believe that the time-limited feature is a natural requirement in delegation. This is similar to the temporary delegations in [1]. However, our solution just requires a global clock and delegating a transformation key for the new time period to the proxy, instead of assuming all users can access an authenticated copy of the new global parameter. Utilizing a recent forward-secure signature scheme [4], which can be seen as a homomorphic compartment signature, we can get constant size forward-secure proxy re-signatures.

References

1. Ateniese, G., Hohenberger, S.: Proxy Re-signatures: New Definitions, Algorithms, and Applications. In: ACM Conference on Computer and Communications Security, pp. 310–319 (2005)
2. Blaze, M., Bleumer, G., Strauss, M.: Divertible Protocols and Atomic Proxy Cryptography. In: Nyberg, K. (ed.) EUROCRYPT 1998. LNCS, vol. 1403, pp. 127–144. Springer, Heidelberg (1998)
3. Boneh, D., Boyen, X., Goh, E.-J.: Hierarchical Identity Based Encryption with Constant Size Ciphertext. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 440–456. Springer, Heidelberg (2005)

4. Boyen, X., Shacham, H., Shen, E., Waters, B.: Forward-Secure Signatures with Untrusted Update. In: ACM Conference on Computer and Communications Security, pp. 191–200. ACM, New York (2006)
5. Boyen, X., Waters, B.: Compact Group Signatures Without Random Oracles. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 427–444. Springer, Heidelberg (2006)
6. Canetti, R., Halevi, S., Katz, J.: A Forward-Secure Public-Key Encryption Scheme. Journal of Cryptology 20(3) (2007)
7. Gentry, C., Silverberg, A.: Hierarchical ID-Based Cryptography. In: Zheng, Y. (ed.) ASIACRYPT 2002. LNCS, vol. 2501, pp. 548–566. Springer, Heidelberg (2002)
8. Goldwasser, S., Micali, S., Rivest, R.L.: A Digital Signature Scheme Secure against Adaptive Chosen-Message Attacks. SIAM Journal of Computing 17(2), 281–308 (1988)
9. Johnson, R., Molnar, D., Song, D.X., Wagner, D.: Homomorphic Signature Schemes. In: Preneel, B. (ed.) CT-RSA 2002. LNCS, vol. 2271, pp. 244–262. Springer, Heidelberg (2002)
10. Shao, J., Cao, Z., Wang, L., Liang, X.: Proxy Re-signature Schemes Without Random Oracles. In: Srinathan, K., Rangan, C.P., Yung, M. (eds.) INDOCRYPT 2007. LNCS, vol. 4859, pp. 197–209. Springer, Heidelberg (2007)

An RSA-Based (t, n) Threshold Proxy Signature Scheme without Any Trusted Combiner

Pei-yih Ting[1] and Xiao-Wei Huang[1,2]

[1] Department of Computer Science and Engineering,
National Taiwan Ocean University, Taiwan, R.O.C.
{pyting,m94570010}@mail.ntou.edu.tw
[2] Institute of Information Science, Academia Sinica, Taiwan, R.O.C.
xwhuang@iis.sinica.edu.tw

Abstract. In this paper we propose and analyze a new RSA-based proxy signature scheme and its corresponding (t, n) threshold scheme. Unlike numerous previous research works, the threshold proxy scheme does not require any trusted combiner and is thus a truly practical approach. The security of both schemes is based on a weaker version of the RSA assumption. Both schemes are unforgeable and especially the threshold proxy scheme inherits the merit of its predecessor - Shoup's RSA threshold scheme - and thus is secure under a multi-party computation setup with active adversaries.

Keywords: Cryptography, Proxy signature, Threshold proxy signature.

1 Introduction

In 2003, Hwang et al. [7] introduced a practical (t, n) threshold proxy signature scheme based on the RSA cryptosystem. Wang et al. [14] found the security flaws of Hwang's scheme the next year. Recently, Kuo et al. [8] proposed a modified (t, n) threshold proxy signature scheme based on the RSA cryptosystem and Chang et al. [2] proposed an RSA-based (t, n) threshold proxy signature scheme with free-will identities. Basically, Hwang's approach leaks $\phi(N)$ through the proxy key and the proxy signature cannot be issued without the knowledge of $\phi(N)$. Thus the follow-up approaches tried to avoid the pitfall by sacrificing some features. For example, both Kuo's and Chang's schemes assumed that there is a trusted combiner in the joint signing process. Now the question is that if there were a trusted combiner in the scheme, why bother using a threshold scheme to distribute the trust? Thus, in this paper we design a new delegation method, which leads to an RSA-based proxy signature scheme without leaking $\phi(N)$. Furthermore, the requirement of trusted parties is eliminated by following the methodology of Shoup's RSA threshold signature [13].

The rest of this paper is organized as follows: we first present the new RSA-based proxy signature scheme in Section 2 and extend it to a threshold proxy scheme in Section 3. The underlying assumption and various security issues of the proposed schemes are analyzed in Section 4. Section 5 gives a brief summary.

T.-C. Wu et al. (Eds.): ISC 2008, LNCS 5222, pp. 277–284, 2008.

2 An RSA-Based Proxy Signature Scheme

We first sketch the RSA-based proxy signature scheme over a variant of RSA signature scheme.

2.1 Setup

In this scheme, there are an original signer P_0 with the RSA private key d_0 and public key (e_0, n_0), such that $gcd(e_0, \phi(n_0)) = 1$ and $e_0 \cdot d_0 = 1 \pmod{n_0}$, a proxy signer P_1 with RSA private key d_1 and public key (e_1, n_1), and a verifier. Let w be the warrant for the proxy signer P_1 and $h(\cdot)$ be a collision-resistant hash function.

2.2 The Proxy Generation Protocol

1. P_0 picks $a \in_R Z_{\phi(\phi(n_0))}$ and computes
   Proxy signature signing key: $D = {d_0}^a \pmod{\phi(n_0)}$,
   Proxy signature verification key: $\begin{cases} E = {e_0}^{h(w)} \pmod{\phi(n_0)} \\ C = h(w)^G \pmod{n_0} \end{cases}$
   where $\in_R$ denotes "select randomly in" and $G = (DE)^{-1} \pmod{\phi(n_0)}$ is a secret of P_0.
2. P_0 publishes $\{E, C, w, \sigma_w = h(E||C||w)^{d_0} \pmod{n_0}\}$ and sends D to P_1 in a secure manner (e.g. let it be encrypted by P_1's public key with suitable reblocking mechanism).
3. P_1 receives the proxy key D and verifies its validity by $C^{ED} = h(w) \pmod{n_0}$.

2.3 The Proxy Signature Signing Protocol

The proxy signer P_1 signs message m for the original signer P_0 as follows:

P_1 computes

$$S = C^{D \cdot h(m)} \pmod{n_0},$$

and signs S with his private key, i.e., $\sigma_1 = h(S)^{d_1} \pmod{n_1}$ with suitable reblocking mechanism (e.g. if $h(S) > n_1$, P_1 can divide $h(S)$ into several small parts) to ensure that the original signer can not forge this proxy signature. Then P_1 sends S and σ_1 to the verifier.

2.4 The Proxy Signature Verification Protocol

A verifier verifies the signature S and σ_1 as follows:

1. He checks that
   $$(\sigma_w)^{e_0} = h(E||C||w) \pmod{n_0}.$$
2. He checks that
   $$S^E = h(w)^{h(m)} \pmod{n_0}.$$
3. He checks that
   $$\sigma_1^{e_1} = h(S) \pmod{n_1}$$
   to ensure that the original signer cannot forge the proxy signature.

The security of this proxy scheme will be analyzed in Section 4. In the next section, we extend the above scheme to a threshold proxy signature scheme by following Shoup's [13] threshold RSA signature scheme, which is based on Shamir's polynomial secret sharing [12].

3 RSA Threshold (t,n) Proxy Signature Scheme

3.1 Setup

We consider an original signer P_0, n proxy signers $P_1, \cdots, P_n$, and a verifier in this scheme. Let d_i be the RSA private key of $P_i, i = 1, \ldots, n$ and (e_i, n_i) be the corresponding public key such that $gcd(e_i, \phi(n_i)) = 1$ and $e_i \cdot d_i = 1 \pmod{\phi(n_i)}$. Let w be the warrant for the proxy signers $\{P_1, \cdots, P_n\}$, t be the threshold, and $h(\cdot)$ be a collision-resistant hash function. Let (e_0, n_0) be the RSA public key of P_0, where $n_0 = p_0 q_0$, p_0 and q_0 are large primes, $p_0 = 2p_0' + 1, q_0 = 2q_0' + 1$, and p_0' and q_0' are large primes also. Let m_0 denote $p_0' q_0'$. e_0 is chosen such that $\gcd(e_0, \phi(n_0)) = 1$, $E = e_0^{h(w)} \pmod{\phi(n_0)}$ is a prime and $E > n$. Let d_0 be the RSA private key of P_0 such that $d_0 \cdot e_0 = 1 \pmod{\phi(n_0)}$.

3.2 The Proxy Sharing Protocol

1. P_0 picks $a \in_R Z_{\phi(m_0)}$ and computes
   Proxy key: $D = {d_0}^a \pmod{m_0}$,
   Proxy signature verification key: $\begin{cases} E = {e_0}^{h(w)} \pmod{\phi(n0)} \\ C = h(w)^G \pmod{n0} \end{cases} (= {e_0}^{h(w)} \pmod{m_0})$
   where $G = (DE)^{-1} \pmod{m_0}$ is a secret of P_0. Note that D is not the proxy signature signing key but the modulo m_0 part of the proxy signature signing key such that we can perform correctly the sharing and signing protocol in the cyclic subgroup Q_{n_0} of quadratic residues in $Z_{n_0}^*$, i.e. $\exists x \in Z_{n_0}^*$ s.t. $v_0 = x^2 \pmod{n_0}$.
2. P_0 secretly picks a polynomial $f(x) = D + r_1 x^1 + \cdots + r_{t-1} x^{t-1} \pmod{m_0}$, where $\{r_i\}_{i=1,\cdots,t-1}$ are random numbers randomly selected from $\{0, \cdots, m_0 - 1\}$.
3. P_0 publishes $\{E, C, w, \sigma_w = h(E||C||w)^{d_0} \pmod{n_0}\}$ and sends $D_i = f(i) \pmod{m_0}$ to P_i in a secure manner for $i = 1, \cdots, n$. The number D_i is the share of the proxy key D for P_i.
4. Each P_i receives his share D_i and verifies its validity jointly by the execution of the proxy signature signing protocol (Section 3.3) and the proxy signature verification protocol (3.4) with an arbitrary message m.
5. P_0 picks a random v_0 in Q_{n_0}, computes and publishes share verification keys $v_i = {v_0}^{\Delta D_i} \pmod{n_0}$ for $i = 1, \cdots, n$, where $\Delta = n!$. The share verification keys $\{v_i\}_{i=0,\cdots,n}$ are used to guarantee the robustness of the entire threshold proxy protocol.

3.3 The Proxy Signature Signing Protocol

t or more proxy signers jointly sign a message m according to the following:

1. Each participating P_i computes and broadcasts the partial proxy signature
$$S_i = \left(C^{h(m)}\right)^{2\Delta D_i} \pmod{n_0}$$
and signs S_i with his private key, i.e., $\sigma_i = h(S_i)^{d_i} \pmod{n_i}$ with suitable reblocking mechanism to ensure that the original signer can not forge this partial proxy signature.
2. Each P_i provides an "equal discrete log" NIZKP [3,13] of $\mathrm{dlog}_{C^{4h(m)\Delta}}(S_i)^2 = \mathrm{dlog}_{v_0^{\Delta}}(v_i)\,(= D_i)$ to ensure the correctness of the partial proxy signature S_i.
3. Everyone can verify the validity of S_i by checking the NIZKP provided by P_i. If there are less than t honest P_is remaining, the signing protocol aborts. Let T denote the index set of valid partial proxy signatures, then, we can move a step toward reconstructing the proxy signature S by calculation $\bar{S}$ as follows:
$$\bar{S} = \prod_{i\in T} S_i^{2\Delta L_i}$$
$$\left(= \prod_{i\in T} C^{h(m)4\Delta D_i \Delta L_i} = C^{h(m)4\sum_{i\in T} D_i \Delta^2 L_i} = C^{h(m)4D\Delta^2}\right) \pmod{n_0},$$
where L_i is the Lagrange coefficient [1]. ΔL_i is guaranteed to be an integer and can be calculated without knowing $\phi(n_0)$. This removes the trusted combiner in Kuo's [8] approach.
4. Since e_0 is chosen such that E is a prime number larger than n, we have $\gcd(4\Delta^2, E) = 1$. Using the extended Euclidean algorithm we can find two integers $\tilde{a}, \tilde{b}$ such that $\tilde{a}4\Delta^2 + \tilde{b}E = 1$. Then the proxy signature can be calculated as
$$S = \bar{S}^{\tilde{a}} h(w)^{h(m)\tilde{b}} \left(= S^{4\Delta^2\tilde{a}} \cdot S^{E\tilde{b}}\right) \pmod{n_0}.$$

3.4 The Proxy Signature Verification Protocol

Anyone can verify the proxy signature $(S, \{\sigma_i\}_{i\in T})$ of the message m with respect to the public values (n_0, e_0), $\{(n_i, e_i)\}_{i\in T}$, and (E, C, w, σ_w) as follows:

1. He checks that
$$(\sigma_w)^{e_0} = h(E||C||w) \pmod{n_0}.$$
2. He checks that
$$S^E = h(w)^{h(m)} \pmod{n_0}.$$
3. In case that the original signer denies his responsibility by blaming the proxy signers, the proxy signers need to check that
$$\sigma_i^{e_i} = h(S_i) \pmod{n_i} \text{ for all } i \in T$$
to ensure that the original signer cannot forge the partial proxy signature.

[1] The Lagrange coefficient is defined as $L_i = \prod_{j\in T, j\neq i} \frac{-j}{i-j}$.

4 Security Analysis

In this section, we review the RSA assumption and show that the proposed proxy signature is unforgeable with the standard RSA assumption and satisfies the properties mentioned in [7].

The RSA assumption [11]: *Given an RSA public key* (n_0, e_0) *and a ciphertext* $c = m^{e_0} \pmod{n_0}$*, it is hard to compute the plaintext* m *without the RSA private key.*

We now show that the "RSA assumption" implies the "composite-exponent RSA assumption", which the security of the proposed scheme is based on.

The composite-exponent RSA assumption: *Given an RSA public key* (n_0, e_0)*, two integer factors* E*,* D*, i.e.* $e_0 = ED$*, and a ciphertext* $c = m^{e_0} \pmod{n_0}$*, it is hard to compute the plaintext* m *without the RSA private key.*

Theorem 1. *The "RSA assumption" implies the "composite-exponent RSA assumption".*

Proof. Assume that there exists a PPT algorithm $A(n_0, E, D, c)$ that breaks the "composite-exponent RSA assumption" with non-negligible probability, we can easily construct a PPT algorithm $A^*(n_0, e_0, c)$ which uses $A(n_0, E, D, c)$ as a blackbox tool to break the "RSA assumption" with non-negligible probability as follows:

$A^*(n_0, e_0, c)$:

1. A^* randomly picks an integer D and computes
$$c' = c^D = m^{e_0 D} \pmod{n_0}$$
Note that the probability of $\gcd(e_0 D, \phi(n_0)) = 1$ is non-negligible although $\phi(n_0)$ is unknown to A^*.
2. A^* invokes $A(n_0, e_0, D, c')$ to get the plaintext m such that $c' = m^{e_0 D} \pmod{n_0}$.
3. A^* checks that
if $c = m^{e_0} \pmod{n_0}$, then ouput m.
if $c \neq m^{e_0} \pmod{n_0}$, then goto 1.

□

Thus the security properties of the proposed scheme can be set up under the "composite-exponent RSA assumption", which is weaker than a standard "RSA assumption".

4.1 Secrecy: The Original Signer's Private Key Must Be Kept Secret

As shown in Section 3.2, P_0's secrets $\phi(n_0)$ and d_0 are protected by G. Even if the proxy D is retrieved (by controlling all proxy signers), the adversary knows

only $DE\left(=G^{-1}\right) \pmod{m_0}$ and cannot get the multiple of m_0 or $\phi(n_0)$ without knowing G. Moreover, given n_0, E, D, w, and C, the adversary cannot derive G due to the "composite-exponent RSA assumption", where (n_0, ED) is the RSA public key. Let y be the RSA signature satisfying $y^{ED} = h(w) \pmod{n_0}$. Note that since $C = h(w)^G \pmod{n_0}$ and $GED = 1 \pmod{m_0}$, we have $\left(y^{ED}\right)^{2G} = y^2 = (h(w))^{2G} = C^2 \pmod{n_0}$. Therefore, we can solve the signature of $h(w)$ by finding integers a and b satisfying $aED + b2 = 1$ and calculating $y = \left(y^{ED}\right)^a \left(y^2\right) = h(w)^a C^{2b} \pmod{n_0}$. However, in general the "composite-exponent RSA assumption" guarantees that it would be hard to find the signature for arbitrary message without the knowledge of G. Furthermore, we know that to recover the private information G is even harder than breaking any RSA problem. Therefore, we conclude that even all proxy signers collaborate there is no way to derive the original signer P_0's private key d_0.

4.2 Proxy Protected: Only the Delegated Proxy Signer Can Generate the Partial Proxy Signature

In the proxy signature signing protocol, the signature σ_i of S_i by P_i, i.e. $h(S_i)^{d_i} \pmod{n_i}$ ensures that the partial proxy signature (S_i, σ_i) can only be generated by the proxy signer P_i. Although the original signer P_0 can compute S_i, he cannot generate the signature σ_i for S_i without knowing P_i's private key d_i. Hence, the proxy signer is protected in the proposed scheme.

4.3 Unforgeability: Only t or More Proxy Signers Can Jointly Generate a Valid Proxy Signature

Since each proxy signer has only a secret share $D_i = f(i)$ of the polynomial f, only t or more proxy signers can cooperatively reconstruct the proxy key D by the Shamir's polynomial sharing secret sharing scheme. The confidentiality of each secret share D_i in taking the proxy sharing protocol and the proxy signature signing protocol can be proved in a secure multi-party computation scenario similar to the one shown by Shoup [13]. Furthermore, the existential unforgeability under no message attack in the random oracle model of the proposed scheme can be proven in a similar way just like the appendix B in [1] under the RSA assumption.

If an adversary is given the signing oracle, the adversary can get two signatures $\sigma = (S||\{\sigma_i\}_{i\in T})$ satisfying $S^E = h(w)^{h(m)} \pmod{n_0}$ and $\sigma_i^{e_i} = h(S_i) \pmod{n_i}$ for all $i \in T$ and $\sigma' = (S'||\{\sigma_i{}'\}_{i\in T'})$ satisfying $S'^E = h(w)^{h(m')} \pmod{n_0}$ and $\sigma_i{}'^{e_i} = h(S_i{}') \pmod{n_i}$ for all $i \in T'$. The adversary can forge S'' such that $S''^E = h(w)^{h(m'')} \pmod{n_0}$ for arbitrary message m'' if $\gcd(h(m), h(m')) = 1$. Since $\gcd(h(m), h(m')) = 1$, there exist a, b such that $ah(m) + bh(m') = 1$ and the adversary can calculate $S'' = (S^a S'^b)^{h(m'')} \pmod{n_0}$ satisfying $S''^E = h(w)^{h(m'')} \pmod{n_0}$. It appears that only the proxy-protected signature scheme is secure under chosen message attacks provided that $\{\sigma_i\}_{i\in T}$ are secure under chosen message attacks. To further limit the potential dangers of chosen message attacks, we can invoke the constructions introduced in chapter 6 of [6].

4.4 Non-repudiation: Any Valid Proxy Signature Must Be Generated by t or More Proxy Signers and the Original Signer Cannot Deny Having Delegated the Power to the Proxy Signers

First, any valid proxy signature must be generated by someone who knows the full proxy key D due to the unforgeability. Second, it requires t or more proxy signers to reconstruct the proxy key D and the proxy signer P_i must also provide σ_i for S_i. Hence, the group of proxy signers in T cannot deny having signed the message m for the original signer P_0. Then we focus on the non-repudiation of the delegation of the original signer. Since the verifier checks that ${\sigma_w}^{e_0} = h(E||C||w) \pmod{n_0}$ with the original signer P_0's public key e_0 in the proxy signature verification phase and the warrant w describes all the information about delegation, the original signer cannot deny having delegated the power to the proxy signers.

4.5 Time Constraint: The Proxy Signature Signing Keys Can Be Used Only During the Delegation Period

In the proposed scheme, the validity period of the delegation should be recorded in the warrant w and the verifier can check whether the delegation is still effective or not in the proxy signature verification protocol. If the delegation has expired, the verifier declares that the proxy signature is invalid. In general, a proxy signature should be appended by a formal time stamp so that any one can verify the effectiveness of the proxy signature signing key even after the validity period of delegation.

4.6 Known Signers: The System can Identify the Actual Signers in the Proxy Group for Internal Auditing

In the proposed scheme, everyone can verify the partial proxy signature (S_i, σ_i) broadcasted by a proxy signer P_i with P_i's public key (e_i, n_i) for auditing purposes. Hence, everyone knows the identities of the actual proxy signers who generate a particular proxy signature and each proxy signer is forced to hold responsibility by this mechanism.

5 Conclusions

We propose an RSA-based (t, n) threshold proxy signature scheme in this paper. The proposed scheme is based on the RSA assumption and the trusted combiner of previous researches is completely eliminated. The resulting scheme is secure and is more flexible and practical than the previous schemes. Although we do not include the security proof for the overall scheme in this paper, the proof is largely a repetition of the works by Bellare [1] and by Shoup [13] with suitable modifications for the proxy signature scenario.

References

1. Bellare, M., Rogaway, P.: Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In: Proceedings of the First ACM Conference on Computer and Communications Security (1993)
2. Chang, Y.F., Chang, C.C.: An RSA-based (t, n) Threshold Proxy Signature Scheme with Free-will Identities. International Journal of Information and Computer Security 1(1/2), 201–209 (2007)
3. Chaum, D., Pedersen, T.: Wallet Databases with Observers. In: Brickell, E.F. (ed.) CRYPTO 1992. LNCS, vol. 740, pp. 89–105. Springer, Heidelberg (1993)
4. Damgård, I., Jurik, M.: A generalisation, a simplification and some applications of paillier's probabilistic public-key system. In: Proceedings of The International Workshop on Practice and Theory in Public Key Cryptography (PKC). LNCS, vol. 1992, pp. 119–136 (2001)
5. Damgård, I., Koprowski, M.: Practical Threshold RSA Signatures Without a Trusted Dealer. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 152–165. Springer, Heidelberg (2001)
6. Goldreich, O.: Foundations of cryptography - a primer. Foundations and Trends in Theoretical Computer Science 1(1), 1–116 (2005)
7. Hwang, M.S., Lu, J.L., Lin, I.C.: A practical (t, n) threshold proxy signature scheme based on the RSA cryptosystem. IEEE Transactions on Knowledge and Data Engineering 15(6) (2003)
8. Kuo, W.C., Chen, M.Y.: A Modified (t, n) Threshold Proxy Signature Scheme based on the RSA cryptosystem. In: Proceedings of the Third International Conference on Information Technology and Applications (ICITA) (2005)
9. Mambo, M., Usuda, K., Okmamoto, E.: Proxy Signatures for Delegating Signing Operation. In: Proceedings of 3rd ACM Conference Computer and Communication Security, pp. 48–57 (1996)
10. Mambo, M., Usuda, K., Okmamoto, E.: Proxy Signatures: Delegation of the Power to Sign Messages. IEICE Transactions on Fundamentals E79-A(9), 1338–1353 (1996)
11. Rivest, R.L., Kaliski, B.: RSA Problem (2003), theory.lcs.mit.edu/~rivest/RivestKaliski-RSAProblem.pdf
12. Shamir, A.: How to share a secret. Communications of the ACM 22, 612–613 (1979)
13. Shoup, V.: Practical Threshold Signatures. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 207–220. Springer, Heidelberg (2000)
14. Wang, G., Bao, F., Zhou, J., Deng, R.H., Lin, I.C.: Comments on "A practical (t, n) Threshold Proxy Signature Scheme based on the RSA cryptosystem. IEEE Transactions on Knowledge and Data Engineering, 16(10) (2004)

Certificate-Based Signature Schemes without Pairings or Random Oracles

Joseph K. Liu[1], Joonsang Baek[1], Willy Susilo[2], and Jianying Zhou[1]

[1] Cryptography and Security Department
Institute for Infocomm Research, Singapore
{ksliu,jsbaek,jyzhou}@i2r.a-star.edu.sg
[2] Centre for Computer and Information Security (CCISR)
School of Computer Science and Software Engineering
University of Wollongong, Australia
wsusilo@uow.edu.au

Abstract. In this paper, we propose two new certificate-based signature (CBS) schemes with new features and advantages. The first one is very efficient as it does not require any pairing computation and its security can be proven using Discrete Logarithm assumption in the random oracle model. We also propose another scheme whose security can be proven in the standard model without random oracles. To the best of our knowledge, these are the *first* CBS schemes in the literature that have such kind of features.

1 Introduction

Public Key Infrastructure (PKI). In a traditional public key cryptography (PKC), a user Alice signs a message using her private key. A verifier Bob verifies the signature using Alice's public key. However, the public key is just merely a random string and it does not provide authentication of the signer by itself. This problem can be solved by incorporating a certificate generated by a trusted party called the Certificate Authority (CA) that provides an unforgeable signature and trusted link between the public key and the identity of the signer. The hierarchical framework is called the public key infrastructure (PKI), which is responsible to issue and manage the certificates (chain). In this case, prior to the verification of a signature, Bob needs to obtain Alice's certificate in advance and verify the validity of her certificate. If it is valid, Bob extracts the corresponding public key which is then used to verify the signature. In the point of view of a verifier, it takes two verification steps for independent signatures. This approach seems inefficient, in particular when the number of users is very large.

Identity-Based cryptography (IBC). Identity-based cryptography (IBC), invented by Shamir [1] in 1984, solves the aforementioned problem by using Alice's identity (or email address) which is an arbitrary string as her public key while the corresponding private key is a result of some mathematical operation that takes as input the user's identity and the master secret key of a trusted

T.-C. Wu et al. (Eds.): ISC 2008, LNCS 5222, pp. 285–297, 2008.

authority, referred to as "Private Key Generator (PKG)". This way, the certificate is implicitly provided and the explicit authentication of public keys is no longer required. The main disadvantage of identity-based cryptography is an unconditional trust to the PKG. Hence, the PKG can impersonate any user, or decrypt any ciphertexts. Hence, IBC is only suitable for a closed organization where the PKG is completely trusted by everyone in the group.

Certificate-Based cryptography (CBC). To integrate the merits of IBC into PKI, Gentry [2] introduced the concept of certificate-based encryption (CBE). A CBE scheme combines a public key encryption scheme and an identity based encryption scheme between a certifier and a user. Each user generates his/her own private and public keys and requests a certificate from the CA while the CA uses the key generation algorithm of an identity based encryption (IBE) [3] scheme to generate the certificate. The certificate is implicitly used as part of the user decryption key, which is composed of the user-generated private key and the certificate. Although the CA knows the certificate, it does not have the user private key. Thus it cannot decrypt any ciphertexts. In addition to CBE, the notion of certificate-based signature (CBS) was first suggested by Kang *et al.* [4]. However, one of their proposed schemes was found insecure against key replacement attack, as pointed out by Li *et al.* [5]. They also proposed a concrete scheme. In parallel to their construction, Au *et al.* [6] proposed a certificate-based ring signature scheme. Nevertheless, all the above schemes require pairing operations and can be only proven in the random oracle model. To date, it is unknown whether the existence of pairing based cryptography is essential for the construction of CBS schemes.

We remark that certificateless cryptography [7] is another stream of research, which is to solve the key escrow problem inherited by IBC. Although at the first glance certificateless cryptography shares several similarities with certificate-based cryptography, each notion has its own merit and distinct features.

Our Contributions. In this paper, we propose two CBS schemes. The first scheme does not require any pairing operations, which is regarded as costly operations compared to other operations such as exponentiation. According to the current MIRACL implementation [8], a 512-bit Tate pairing takes 20 ms whereas a 1024-bit prime modular exponentiation takes 8.8 ms. Without pairing, our scheme is more efficient than all of the previous schemes proposed so far [4,5,6]. This distinct and interesting property enables our scheme to be implemented in some power-constrained devices, such as wireless sensor networks.

In addition, our second scheme can be proven in the standard model without random oracles. All previous schemes mentioned above rely on the random oracle model to prove their security. It is generally believed that cryptographic schemes relaying on the random oracles may not be secure if the underlying random oracles are realized as hash functions in the real world. For some applications that require very high security, it is believed that only those schemes that can be proven in the standard model without random oracles must be employed. Hence our second scheme can be suitable for those particular applications.

2 Preliminaries

2.1 Notations

Pairing. Let e be a bilinear map such that $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ such that it has the following properties:

- $\mathbb{G}$ and $\mathbb{G}_T$ are cyclic multiplicative groups of prime order p.
- each element of $\mathbb{G}$ and $\mathbb{G}_T$ has unique binary representation.
- g is a generator of $\mathbb{G}$.
- (Bilinear) $\forall x, y \in \mathbb{G}$ and a, $b \in \mathbb{Z}_p$, $e(x^a, y^b) = e(x, y)^{ab}$.
- (Non-degenerate) $e(g, g) \neq 1$.

2.2 Mathematical Assumptions

Definition 1 (Discrete Logarithm (DL) Assumption). *Given a group G of prime order q with generator g and elements $A \in G$, the DL problem in G is to output $\alpha \in \mathbb{Z}_q$ such that $A = g^\alpha$.*

An adversary $\mathcal{B}$ has at least an ϵ advantage if

$$\Pr[\mathcal{B}(g, A) = \alpha \mid A = g^\alpha] \geq \epsilon$$

We say that the (ϵ, t)-DL assumption holds in a group G if no algorithm running in time at most t can solve that DL problem in G with advantage at least ϵ.

Definition 2 (Generalized Computational Diffie-Hellman (GCDH) Assumption). *Given a group G of prime order p with generator g and elements $g^a, g^b \in G$ where a, b are selected uniformly at random from $\mathbb{Z}_p^*$, the GCDH problem in G is to output (g^{abc}, g^c), where $g^c \in G$.*

An adversary $\mathcal{B}$ has at least an ϵ advantage if

$$\Pr[\mathcal{B}(g, g^a, g^b) = (g^{abc}, g^c)] \geq \epsilon$$

We say that the (ϵ, t)-GCDH assumption holds in a group G if no algorithm running in time at most t can solve the GCDH problem in G with advantage at least ϵ.

It is a strictly weaker assumption than the Generalized Bilinear Diffie-Hellman (GBDH) assumption [7], which is defined as follow:

Definition 3 (Generalized Bilinear Diffie-Hellman (GBDH) Assumption). *Given a group G of prime order p with generator g and elements $g^a, g^b, g^{b'} \in G$ where a, b, b' are selected uniformly at random from $\mathbb{Z}_p^*$, the GBDH problem in G is to output $(e(g^c, g)^{abb'}, g^c)$, where $g^c \in G$.*

An adversary $\mathcal{B}$ has at least an ϵ advantage if

$$\Pr[\mathcal{B}(g, g^a, g^b, g^{b'}) = (e(g^c, g)^{abb'}, g^c)] \geq \epsilon$$

We say that the (ϵ, t)-GBDH assumption holds in a group G if no algorithm running in time at most t can solve the GBDH problem in G with advantage at least ϵ.

Lemma 1. *The GCDH assumption is strictly weaker than the GBDH assumption.*

Proof. Assume there is a GCDH adversary $\mathcal{A}$. We construct another adversary $\mathcal{B}$ to solve the GBDH problem. Given $(g, g^a, g^b, g^{b'})$ as the GBDH problem instance, $\mathcal{B}$ gives (g, g^a, g^b) to $\mathcal{A}$. $\mathcal{A}$ outputs (g^{abc}, g^c) with probability ϵ and time t. $\mathcal{B}$ outputs $(e(g^{b'}, g^{abc}), g^c)$ as the solution to the GBDH problem. Since $e(g^{b'}, g^{abc}) = e(g^c, g)^{abb'}$, it is a valid solution. Thus $\mathcal{B}$ can solve GBDH problem with time t and probability ϵ.

Definition 4 (Many-DH Assumption [9] (Simplified Version) [1]). *Given a group G of prime order p with generator g and elements $g^a, g^b, g^c, g^{ab}, g^{ac}, g^{bc} \in G$ where a, b, c are selected uniformly at random from $\mathbb{Z}_p^*$, the Many-DH problem in G is to output g^{abc}.*

An adversary $\mathcal{B}$ has at least an ϵ advantage if

$$\Pr[\mathcal{B}(g, g^a, g^b, g^c, g^{ab}, g^{ac}, g^{bc}) = g^{abc}] \geq \epsilon$$

We say that the (ϵ, t)-Many-DH assumption holds in a group G if no algorithm running in time at most t can solve the Many-DH problem in G with advantage at least ϵ.

3 Security Model

Definition 5. *A certificate-based signature (CBS) scheme is defined by six algorithms:*

- *Setup is a probabilistic algorithm taking as input a security parameter. It returns the certifier's master key msk and public parameters $param$. Usually this algorithm is run by the CA.*
- *UserKeyGen is a probabilistic algorithm that takes $param$ as input. When run by a client, it returns a public key PK and a secret key usk.*
- *Certify is a probabilistic algorithm that takes as input $(msk, \tau, param, PK, ID)$ where ID is a binary string representing the user information. It returns $Cert'_\tau$ which is sent to the client. Here τ is a string identifying a time period.*
- *Consolidate is a deterministic certificate consolidation algorithm taking as input $(param, \tau, Cert'_\tau)$ and optionally $Cert_{\tau-1}$. It returns $Cert_\tau$, the certificate used by a client in time period τ.*
- *Sign is a probabilistic algorithm taking as input $(\tau, param, m, Cert_\tau, usk)$ where m is a message. It outputs a ciphertext σ.*
- *Verify is a deterministic algorithm taking $(param, PK, ID, \sigma)$ as input in time period τ. It returns either valid indicating a valid signature, or the special symbol $\perp$ indicating invalid.*

[1] In the original version presented in [9], the number of input tuples can be as much as $\mathcal{O}(\log k)$ for some security parameter k. Here we simplify it for just enough to our scheme.

We require that if σ is the result of applying algorithm Sign *with intput $(\tau, param, m, Cert_\tau, usk)$ and (usk, PK) is a valid key-pair, then* valid *is the result of applying algorithm* Verify *on input $(param, PK, ID, \sigma)$, where $Cert_\tau$ is the output of* Certify *and* Consolidate *algorithms on input $(msk, param, \tau, PK)$. That is, we have*

$$\mathsf{Verify}_{PK,ID}(\mathsf{Sign}_{\tau,Cert_\tau,usk}(m)) = \mathsf{valid}$$

We also note that a concrete CBS scheme may not involve certificate consolidation. In this situation, algorithm Consolidate *will simply output $Cert_\tau = Cert'_\tau$.*

In the rest of this paper, for simplicity, we will omit Consolidate and the time identifying string τ in all notations.

The security of CBS is defined by two different games and the adversary chooses which game to play. In Game 1, the adversary models an uncertified entity while in Game 2, the adversary models the certifier in possession of the master key msk attacking a fixed entity's public key. We use the enhanced model by Li *et al.* [5] which captures key replacement attack in the security of Game 1.

Definition 6 (CBS Game 1 Existential Unforgeability). *The challenger runs* Setup*, gives param to the adversary $\mathcal{A}$ and keeps msk to itself. The adversary then interleaves certification and signing queries as follows:*

- *On user-key-gen query (ID), if ID has been already created, nothing is to be carried out. Otherwise, the challenger runs the algorithm* UserKeyGen *to obtains a secret/public key pair (usk_{ID}, PK_{ID}) and adds to the list L. In this case, ID is said to be 'created'. In both cases, PK_{ID} is returned.*
- *On corruption query (ID), the challenger checks the list L. If ID is there, it returns the corresponding secret key usk_{ID}. Otherwise nothing is returned.*
- *On certification query (ID), the challenger runs* Certify *on input $(msk, param, PK, ID)$, where PK is the public key returned from the user-key-gen query, and returns $Cert$.*
- *On signing query (ID, PK, m), the challenger generates σ by using algorithm* Sign*.*

$\mathcal{A}$ can replace any user ID public key with his own choice, but once it has replaced the public key, it cannot obtain the certificate of the false public key from the challenger. Finally $\mathcal{A}$ outputs a signature σ^*, a message m^* and a public key PK^* with user information ID^*. The adversary wins the game if

- *σ^* is a valid signature on the message m^* under the public key PK^* with user information ID^*, where PK^* might not be the one returned from user-key-gen query.*
- *ID^* has never been submitted to the certification query.*
- *(ID^*, PK^*, m^*) has never been submitted to the signing query.*

We define $\mathcal{A}$'s advantage in this game to be $Adv(\mathcal{A}) = \Pr[\mathcal{A}$ wins $]$.

Definition 7 (CBS Game 2 Existential Unforgeability). *The challenger runs* Setup*, gives param and msk to the adversary $\mathcal{A}$. The adversary interleaves user-key-gen queries, corruption queries and signing queries as in Game 1. But different from Game 1, the adversary is not allowed to replace any public key.*

Finally $\mathcal{A}$ outputs a signature σ^*, a message m^* and a public key PK^* with user information ID^*. The adversary wins the game if

- *σ^* is a valid signature on the message m^* under the public key PK^* with user information ID^*.*
- *PK^* is an output from user-key-gen query.*
- *ID^* has never been submitted to corruption query.*
- *(ID^*, PK^*, m^*) has never been submitted to the signing query.*

We define $\mathcal{A}$'s advantage in this game to be $Adv(\mathcal{A}) = \Pr[\mathcal{A}$ wins $]$.

We note that our model does not support security against *Malicious Certifier*. That is, we assume that the certifier generates all public parameters honest, according to the algorithm specified. The adversarial certifier is only given the master secret key, instead of allowing to generate all public parameters. Although malicious certifier has not been discussed in the literature, similar concept of *Malicious Key Generation Centre (KGC)* [10] has been formalized in the area of certificateless cryptography.

4 A CBS without Pairing

Our scheme is motivated from Beth's identification scheme [11].

Setup. Let $\mathbb{G}$ be a multiplicative group with order q. The PKG selects a random generator $g \in \mathbb{G}$ and randomly chooses $x \in_R \mathbb{Z}_q^*$. It sets $X = g^x$. Let $H : \{0,1\}^* \rightarrow \mathbb{Z}_q^*$ be a cryptographic hash function. The public parameters param and master secret key msk are given by

$$\mathsf{param} = (\mathbb{G}, q, g, X, H) \qquad msk = x$$

UserKeyGen. User selects a secret value $u \in \mathbb{Z}_q^*$ as his secret key usk, and computes his public key PK as (g^u, X^u, π_u) where π_u is the following non-interactive proof-of-knowledge (PoK) [2]:

$$PK\{(u) : U_1 = g^u \wedge U_2 = X^u\}$$

Certify. Let $\tilde{h} = H(PK, ID)$ for user with public key PK and binary string ID which is used to identify the user. To generate a certificate for this user, the CA randomly selects $r \in_R \mathbb{Z}_q^*$, computes

$$R = g^r \qquad s = r^{-1}(\tilde{h} - xR) \bmod q$$

[2] For the details of notation and implementation of PoK, please refer to [12].

The certificate is (R, s). Note that a correctly generated certificate should fulfill the following equality:

$$R^s X^R = g^{\tilde{h}} \tag{1}$$

Sign. To sign a message $m \in \{0,1\}^*$, the signer with public key PK (and user info ID), certificate (R, s) and secret key u, randomly selects $y \in_R \mathbb{Z}_q^*$, computes

$$Y = R^{-y} \qquad h = H(Y, R, m) \qquad z = y + h\ s\ u \bmod q$$

and outputs (Y, R, z) as the signature σ.

Verify. Given a signature $\sigma = (Y, R, z)$ for a public key PK on a message m, a verifier first checks whether π_u is a valid PoK. If not, output $\perp$. Otherwise computes $h = H(Y, R, m)$, $\tilde{h} = H(PK, ID)$, and checks whether

$$(g^u)^{h\tilde{h}} \stackrel{?}{=} R^z Y (X^u)^{hR} \tag{2}$$

Output valid if it is equal. Otherwise, output $\perp$.

4.1 Security Analysis

Correctness. It is easy to see that the signature scheme is correct, as shown in following:

$$R^z Y (X^u)^{hR} = R^{y+hsu} R^{-y} X^{uhR} = g^{rhsu} g^{uxhR}$$
$$= g^{rhu\left(r^{-1}(\tilde{h}-xR)\right)+uxhR} = g^{hu\tilde{h}-huxR+uxhR} = (g^u)^{h\tilde{h}}$$

Theorem 1 (Unforgeability against Game 1 Adversary). *The CBS scheme without pairing is (ϵ, t)-existential unforgeable against Game 1 adversary (defined in Section 3) with advantage at most ϵ and runs in time at most t, assuming that the (ϵ', t')-DL assumption holds in $\mathbb{G}$, where*

$$\epsilon' = \left(1 - \frac{q_h(q_e + q_s)}{q}\right)\left(1 - \frac{1}{q}\right)\left(\frac{1}{q_h}\right)\epsilon, \qquad t' = t + \mathcal{O}(q_e + q_s)E$$

and q_e, q_s, q_h are the numbers of certification queries, signing queries and hashing queries the adversary is allowed to make and E is the time for an exponentiation operation.

Proof. Here we follow the idea from [13,14]. Assume there exists a forger $\mathcal{A}$. We construct an algorithm $\mathcal{B}$ that makes use of $\mathcal{A}$ to solve discrete logarithm problem. $\mathcal{B}$ is given a multiplicative group $\mathbb{G}$ with generator g and order q, and a group element $A \in \mathbb{G}$. $\mathcal{B}$ is asked to find $\alpha \in \mathbb{Z}_q$ such that $g^\alpha = A$.

Setup: $\mathcal{B}$ chooses a hash function $H : \{0,1\}^* \to \mathbb{Z}_q$ which behaves like a random oracle. $\mathcal{B}$ is responsible for the simulation of this random oracle. $\mathcal{B}$ assigns $X = A$ and outputs the public parameter $param = (\mathbb{G}, q, g, X, H)$ to $\mathcal{A}$.

User-Key-Gen / Corruption Query: $\mathcal{B}$ generates the secret and public key pair according to the algorithm and stores in the table and outputs the public key. On the corruption query, $\mathcal{B}$ returns the corresponding secret key.

Certification Query: $\mathcal{A}$ is allowed to make certification query for a public key PK with identification string ID. $\mathcal{B}$ simulates the oracle as follow. It randomly chooses $a, b \in_R \mathbb{Z}_q$ and sets

$$R = X^a g^b \qquad s = -a^{-1}R \bmod q \qquad H(PK, ID) = bs \bmod q$$

Note that (R, s) generated in this way satisfies the equation (1) in the Certify algorithm. It is a valid certificate. $\mathcal{B}$ outputs (R, s) as the certificate of PK, ID and store the value of $(R, s, H(PK, ID), PK, ID)$ in the table for consistency. Later if $\mathcal{A}$ queries the H random oracle for PK, ID, $\mathcal{B}$ outputs the same value. If PK, ID is not found in the table, $\mathcal{B}$ executes the certification oracle simulation, stores the value $(R, s, H(PK, ID), PK, ID)$ in the table and outputs $H(PK, ID)$ only.

Signing Query: $\mathcal{A}$ queries the signing oracle for a message m and a public key $PK = (g^u, X^u, \pi_u)$ with identification string ID. $\mathcal{B}$ first checks that whether π_u is a valid PoK for (g^u, X^u). If not , output $\perp$. Else further checks whether PK, ID has been queries for the H random oracle or extraction oracle before. If yes, it just retrieves $(R, s, H(PK, ID), PK, ID)$ from the table. Let $\tilde{h} = H(PK, ID)$. It also randomly generates $h, z \in_R \mathbb{Z}_q$, and sets $Y = \frac{(g^u)^{h\tilde{h}}}{R^z (X^u)^{hR}}$ and assigns the value h to the random oracle query of $H(Y, R, m)$. It outputs the signature (Y, R, z) for the message m and stores the value h, corresponding to $H(Y, R, m)$ in the hash table for consistency. If PK, ID has not been queried to the random oracle or certification oracle, $\mathcal{B}$ executes the simulation of the certification oracle and uses the corresponding certificate to sign the message by the above algorithm.

Output Calculation: Finally the adversary $\mathcal{A}$ outputs a forged signature $\sigma^*_{(1)} = (Y^*, R^*, z^*_{(1)})$ on message m^* and public key PK^* with identification string ID^*. $\mathcal{B}$ rewinds $\mathcal{A}$ to the point it just queries $H(Y^*, R^*, m^*)$ and supplies with a different value (corresponding to the same input value to the hash query). $\mathcal{A}$ outputs another pair of signature $\sigma^*_{(2)} = (Y^*, R^*, z^*_{(2)})$. $\mathcal{B}$ repeats twice and obtains $\sigma^*_{(3)} = (Y^*, R^*, z^*_{(3)})$ and $\sigma^*_{(4)} = (Y^*, R^*, z^*_{(4)})$. Note that Y^* and R^* should be the same every time. We let c_1, c_2, c_3, c_4 be the output of the random oracle queries $H(Y^*, R^*, m^*)$ for the first, second, third and forth time.

We also denote $u, r, x, y \in \mathbb{Z}_q$ such that $g^r = R^*$, $g^x = X$, $g^y = Y^*$ and $PK^* = (g^u, X^u)$. From equation (2), we have

$$c_i u H(PK^*, ID^*) = r z^*_{(i)} + y + x u c_i R^* \bmod q \qquad \text{for } i = 1, 2, 3, 4$$

In these equations, only r, y, x, u are unknown to $\mathcal{B}$. $\mathcal{B}$ solves for these values from the above 4 linear independent equations, and outputs x as the solution of the discrete logarithm problem.

Probability Analysis: The simulation of the random oracle fails if the oracle assignment $H(PK, ID)$ causes inconsistency. It happens with probability at most

q_h/q. Hence the simulation is successful $q_e + q_s$ times (since $H(PK, ID)$ may also be queried in the signing oracle if PK, ID has not been queried in the certification oracle) with probability at least

$$\left(1 - \frac{q_h}{q}\right)^{q_e+q_s} \geq 1 - \frac{q_h(q_e + q_s)}{q}$$

Due to the ideal randomness of the random oracle, there exists a query $H(Y^*, R^*, m^*)$ with probability at least $1 - 1/q$. $\mathcal{B}$ guesses it correctly as the point of rewind, with probability at least $1/q_h$. Thus the overall successful probability is

$$\left(1 - \frac{q_h(q_e + q_s)}{q}\right)\left(1 - \frac{1}{q}\right)\left(\frac{1}{q_h}\right)\epsilon$$

The time complexity of the algorithm $\mathcal{B}$ is dominated by the exponentiations performed in the certification and signing queries, which is equal to $t + \mathcal{O}(q_e + q_s)E$ □

Theorem 2 (Unforgeability against Game 2 Adversary). *The CBS scheme without pairing is (ϵ, t)-existential unforgeable against Game 2 adversary (defined in Section 3) with advantage at most ϵ and runs in time at most t, assuming that the (ϵ', t')-DL assumption holds in $\mathbb{G}$, where*

$$\epsilon' = \left(1 - \frac{q_h q_s}{q}\right)\left(1 - \frac{1}{q}\right)\left(\frac{1}{q_h}\right)\left(\frac{1}{q_u}\right)\epsilon, \qquad t' = t + \mathcal{O}(q_s)E$$

and q_s, q_h, q_u are the numbers of signing queries, hashing queries and user-key-gen queries the adversary is allowed to make and E is the time for an exponentiation operation.

Proof. Assume there exists a forger $\mathcal{A}$. We construct an algorithm $\mathcal{B}$ that makes use of $\mathcal{A}$ to solve discrete logarithm problem. $\mathcal{B}$ is given a multiplicative group $\mathbb{G}$ with generator g and order q, and a group element $A \in \mathbb{G}$. $\mathcal{B}$ is asked to find $\alpha \in \mathbb{Z}_q$ such that $g^\alpha = A$.

Setup: $\mathcal{B}$ chooses a hash function $H : \{0,1\}^* \to \mathbb{Z}_q$ which behaves like a random oracle. $\mathcal{B}$ is responsible for the simulation of this random oracle. $\mathcal{B}$ randomly chooses $x \in_R \mathbb{Z}_q$ and sets $X = g^x$. It outputs the public parameter $param = (\mathbb{G}, q, g, X, H)$ to $\mathcal{A}$.

User-Key-Gen Query: $\mathcal{B}$ chooses a particular query ID' and assign the public key $PK' = (A, A^x, \pi')$ where π' can be simulated by the control of the random oracle. For the other queries, $\mathcal{B}$ generates the secret and public key pair according to the algorithm and stores the value in the table.

Corruption Query: If the query is not ID', $\mathcal{B}$ outputs the corresponding secret key from the table. Otherwise $\mathcal{B}$ aborts.

Signing Query: It can be simulated in the same way as in Game 1, which also does not require the knowledge of the secret key.

Output Calculation: Finally the adversary $\mathcal{A}$ outputs a forged signature $\sigma^*_{(1)} = (Y^*, R^*, z^*_{(1)})$ on message m^* and public key PK^* with identification string ID^*.

If $PK^* \neq PK'$, $\mathcal{B}$ aborts. Otherwise $\mathcal{B}$ rewinds $\mathcal{A}$ to the point it just queries $H(Y^*, R^*, m^*)$ and supplies with a different value (corresponding to the same input value to the hash query). $\mathcal{A}$ outputs another pair of signature $\sigma^*_{(2)} = (Y^*, R^*, z^*_{(2)})$. $\mathcal{B}$ repeats and obtains $\sigma^*_{(3)} = (Y^*, R^*, z^*_{(3)})$. Note that Y^* and R^* should be the same every time. We let c_1, c_2, c_3 be the output of the random oracle queries $H(Y^*, R^*, m^*)$ for the first, second and third.

We also denote $u, r, y \in \mathbb{Z}_q$ such that $g^r = R^*$, $g^y = Y^*$ and $PK^* = (g^u, X^u)$. From equation (2), we have

$$c_i u H(PK^*, ID^*) = r z^*_{(i)} + y + x u c_i R^* \bmod q \qquad \text{for } i = 1, 2, 3$$

In these equations, only r, y, u are unknown to $\mathcal{B}$. $\mathcal{B}$ solves for these values from the above 3 linear independent equations, and outputs u as the solution of the discrete logarithm problem.

Probability Analysis: The simulation of the random oracle fails if the oracle assignment $H(PK, ID)$ causes inconsistency. It happens with probability at most q_h/q. Hence the simulation is successful q_s times with probability at least $\left(1 - \frac{q_h}{q}\right)^{q_s} \geq 1 - \frac{q_h q_s}{q}$ Due to the ideal randomness of the random oracle, there exists a query $H(Y^*, R^*, m^*)$ with probability at least $1 - 1/q$. $\mathcal{B}$ guesses it correctly as the point of rewind, with probability at least $1/q_h$. In addition, $\mathcal{B}$ needs to guess correctly that $PK' = PK^*$, which happens with probability $1/q_u$. Thus the overall successful probability is $\left(1 - \frac{q_h q_s}{q}\right)\left(1 - \frac{1}{q}\right)\left(\frac{1}{q_h}\right)\left(\frac{1}{q_u}\right)\epsilon$ The time complexity of the algorithm $\mathcal{B}$ is dominated by the exponentiations performed in the signing queries, which is equal to $t + \mathcal{O}(q_s)E$ □

5 A CBS without Random Oracles

Our scheme is motivated from the identity-based encryption scheme from Waters [16]. Let $H_u : \{0,1\}^* \rightarrow \{0,1\}^{n_u}$ and $H_m : \{0,1\}^* \rightarrow \{0,1\}^{n_m}$ be two collision-resistant cryptographic hash functions for some $n_u, n_m \in \mathbb{Z}$.

Setup. Select a pairing $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ where the order of $\mathbb{G}$ is p. Let g be a generator of $\mathbb{G}$. Randomly select $\alpha \in_R \mathbb{Z}_p$, $g_2 \in_R \mathbb{G}$ and compute $g_1 = g^\alpha$. Also select randomly the following elements:

$$u', m' \in_R \mathbb{G} \qquad \hat{u}_i \in_R \mathbb{G} \text{ for } i = 1, \ldots, n_u \qquad \hat{m}_i \in_R \mathbb{G} \text{ for } i = 1, \ldots, n_m$$

Let $\hat{U} = \{\hat{u}_i\}, \hat{M} = \{\hat{m}_i\}$. The public parameters param are $(e, \mathbb{G}, \mathbb{G}_T, p, g, g_1, g_2, u', \hat{U}, m', \hat{M})$ and the master secret key msk is g_2^α.

UserKeyGen. User selects a secret value $x \in \mathbb{Z}_p$ as his secret key usk, and computes his public key PK as $(\mathsf{pk}^{(1)}, \mathsf{pk}^{(2)}) = (g^x, g_1^x)$.

Certify. Let $\mathfrak{u} = H_u(PK, ID)$ for user with public key PK and binary string ID which is used to identify the user. Let $\mathfrak{u}[i]$ be the i-th bit of $\mathfrak{u}$. Define $\mathcal{U} \subset \{1, \ldots, n_u\}$ to be the set of indicies such that $\mathfrak{u}[i] = 1$.

To construct the certificate, the CA randomly selects $r_u \in_R \mathbb{Z}_p$ and computes

$$\left(g_2^{\alpha}(U)^{r_u}, g^{r_u}\right) = (\mathsf{cert}^{(1)}, \mathsf{cert}^{(2)}) \qquad \text{where } U = u' \prod_{i \in \mathcal{U}} \hat{u}_i$$

Sign. To sign a message $m \in \{0,1\}^*$, the signer with identity PK (and user information ID) , certificate $(\mathsf{cert}^{(1)}, \mathsf{cert}^{(2)})$ and secret key usk, compute $\mathfrak{m} = H_m(m)$. Let $\mathfrak{m}[i]$ be the i-th bit of $\mathfrak{m}$ and $\mathcal{M} \subset \{1, \ldots, n_m\}$ be the set of indicies i such that $\mathfrak{m}[i] = 1$. Randomly select $r_\pi, r_{\mathfrak{m}} \in_R \mathbb{Z}_p$, compute $\mathfrak{u} = H_u(PK, \mathsf{userinfo})$, $U = u' \prod_{i \in \mathcal{U}} \hat{u}_i$ and

$$\sigma = \left(\left(\mathsf{cert}^{(1)}\right)^{usk} (U)^{r_\pi} \left(m' \prod_{i \in \mathcal{M}} \hat{m}_i\right)^{r_{\mathfrak{m}}}, \left(\mathsf{cert}^{(2)}\right)^{usk} g^{r_\pi}, g^{r_{\mathfrak{m}}} \right) = (V, R_\pi, R_{\mathfrak{m}})$$

Verify. Given a signature $\sigma = (V, R_\pi, R_{\mathfrak{m}})$ for a public key PK and user information ID on a message m, a verifier first checks whether $e(g^x, g_1) = e(g_1^x, g)$. If not, outputs $\perp$. Otherwise computes $\mathfrak{m} = H_m(m)$, $\mathfrak{u} = H_u(PK, ID)$, $U = u' \prod_{i \in \mathcal{U}} \hat{u}_i$ and checks whether

$$e(V, g) \stackrel{?}{=} e(g_2, g_1^x)\ e(U, R_\pi)\ e(m' \prod_{i \in \mathcal{M}} \hat{m}_i, R_{\mathfrak{m}})$$

Output valid if it is equal. Otherwise output $\perp$.

5.1 Security Analysis

Correctness. The correctness of the scheme is as follows.

$$\begin{aligned} e(V, g) &= e\left(g_2^{\alpha x}\ U^{r_u x}\ U^{r_\pi}\ \left(m' \prod_{i \in \mathcal{M}} \hat{m}_i\right)^{r_m},\ g\right) \\ &= e(g_2^x, g^\alpha)\ e(U^{r_u x + r_\pi}, g)\ e\left(\left(m' \prod_{i \in \mathcal{M}} \hat{m}_i\right)^{r_m}, g\right) \\ &= e(g_2, g_1^x)\ e(U, g^{r_u x + r_\pi})\ e(m' \prod_{i \in \mathcal{M}} \hat{m}_i, g^{r_{\mathfrak{m}}}) \\ &= e(g_2, g_1^x)\ e(U, R_\pi)\ e(m' \prod_{i \in \mathcal{M}} \hat{m}_i, R_{\mathfrak{m}}) \end{aligned}$$

Theorem 3 (Unforgeability against Game 1 Adversary). *The CBS scheme without random oracles is (ϵ, t)-existential unforgeable against Game 1 adversary (defined in Section 3) with advantage at most ϵ and runs in time at most t, assuming that the (ϵ', t')-GCDH assumption holds in $\mathbb{G}$, where*

$$\epsilon' \geq \frac{\epsilon}{16(q_e + q_s)(n_u + 1)q_s(n_m + 1)},\ t' = t + O\Big(\big(q_e n_u + q_s(n_u + n_m)\big)\rho + (q_e + q_s)\tau\Big)$$

where q_e is the number of queries made to the Certification Query, q_s is the number of queries made to the Signing Query, and ρ and τ are the time for a multiplication and an exponentiation in $\mathbb{G}$ respectively.

The proof is given in the full version of this paper [15].

Theorem 4 (Unforgeability against Game 2 Adversary). *The CBS scheme without random oracles is (ϵ, t)-existential unforgeable against Game 2 adversary (defined in Section 3) with advantage at most ϵ and runs in time at most t, assuming that the (ϵ', t')-Many-DH assumption holds in $\mathbb{G}$, where*

$$\epsilon' \geq \frac{\epsilon}{16 q_s (n_u + 1) q_s (n_m + 1) q_k} \ , \ t' = t + O\Big(\big(q_s (n_u + n_m)\big)\rho + (q_k + q_s)\tau \Big)$$

where q_s is the number of queries made to the Signing Queries, q_k is the number of queries made to the User-key-gen Queries and ρ and τ are the time for a multiplication and an exponentiation in $\mathbb{G}$ respectively.

The proof is given in the full version of this paper [15].

6 Concluding Remarks

In this paper, we proposed two certificate-based signature schemes. The first one does not require any pairing operations. Thus, it is very efficient and particularly suitable to be implemented in some power-constrained devices, such as wireless sensor networks. The second one does not require random oracles for proving its security. It may be suitable for applications that require a high level of security with regard to the fact that cryptographic schemes using the random oracles may not be secure if the random oracles are replaced by conventional hash functions in reality.

References

1. Shamir, A.: Identity-Based Cryptosystems and Signature Schemes. In: Blakely, G.R., Chaum, D. (eds.) CRYPTO 1984. LNCS, vol. 196, pp. 47–53. Springer, Heidelberg (1985)
2. Gentry, C.: Certificate-based encryption and the certificate revocation problem. In: EUROCRYPT 2003. LNCS, vol. 2656, pp. 272–293. Springer, Heidelberg (2003)
3. Boneh, D., Franklin, M.K.: Identity-Based Encryption from the Weil Pairing. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 213–229. Springer, Heidelberg (2001)
4. Kang, B.G., Park, J.H., Hahn, S.G.: A certificate-based signature scheme. In: Okamoto, T. (ed.) CT-RSA 2004. LNCS, vol. 2964, pp. 99–111. Springer, Heidelberg (2004)
5. Li, J., Huang, X., Mu, Y., Susilo, W., Wu, Q.: Certificate-based signature: Security model and efficient construction. In: López, J., Samarati, P., Ferrer, J.L. (eds.) EuroPKI 2007. LNCS, vol. 4582, pp. 110–125. Springer, Heidelberg (2007)
6. Au, M., Liu, J., Susilo, W., Yuen, T.: Certificate based (linkable) ring signature. In: Dawson, E., Wong, D.S. (eds.) ISPEC 2007. LNCS, vol. 4464, pp. 79–92. Springer, Heidelberg (2007)
7. Al-Riyami, S.S., Paterson, K.: Certificateless public key cryptography. In: Laih, C.-S. (ed.) ASIACRYPT 2003. LNCS, vol. 2894, pp. 452–473. Springer, Heidelberg (2003)

8. MIRACL, http://www.shamus.ie/
9. Lysyanskaya, A.: Unique signatures and verifiable random functions from the DH-DDH separation. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 597–612. Springer, Heidelberg (2002)
10. Au, M., Chen, J., Liu, J., Mu, Y., Wong, D., Yang, G.: Malicious KGC attacks in certificateless cryptography. In: ASIACCS 2007, pp. 302–311. ACM Press, New York (2007)
11. Beth, T.: Efficient Zero-Knowledged Identification Scheme for Smart Cards. In: Günther, C.G. (ed.) EUROCRYPT 1988. LNCS, vol. 330, pp. 77–86. Springer, Heidelberg (1988)
12. Camenisch, J., Stadler, M.: Efficient Group Signature Schemes for Large Groups (Extended Abstract). CRYPTO 1997. In: Kaliski Jr., B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 410–424. Springer, Heidelberg (1997)
13. Bellare, M., Namprempre, C., Neven, G.: Security Proofs for Identity-Based Identification and Signature Schemes. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 268–286. Springer, Heidelberg (2004)
14. Bellare, M., Namprempre, C., Neven, G.: Security Proofs for Identity-Based Identification and Signature Schemes (Full version). Cryptology ePrint Archive, Report 2004/252 (2004), http://eprint.iacr.org/
15. Liu, J., Baek J., Susilo, W., Zhou, J.: Certificate-Based Signature Schemes without Pairings or Random Oracles (Full version). Cryptology ePrint Archive, Report 2008/275 (2008), http://eprint.iacr.org/
16. Waters, B.: Efficient identity-based encryption without random oracles. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 114–127. Springer, Heidelberg (2005)
17. Paterson, K., Schuldt, J.: Efficient identity-based signatures secure in the standard model. In: Batten, L.M., Safavi-Naini, R. (eds.) ACISP 2006. LNCS, vol. 4058, pp. 195–206. Springer, Heidelberg (2006)

Improved Impossible Differential Attacks on Large-Block Rijndael

Lei Zhang[1,2], Wenling Wu[1], Je Hong Park[3],
Bon Wook Koo[3], and Yongjin Yeom[3]

[1] State Key Laboratory of Information Security,
Institute of Software, Chinese Academy of Sciences, Beijing 100190, P.R. China
{zhanglei1015,wwl}@is.iscas.ac.cn
[2] State Key Laboratory of Information Security,
Graduate University of Chinese Academy of Sciences, Beijing 100049, P.R. China
[3] Electronics and Telecommunications Research Institute, Daejeon, Korea
{bwkoo,jhpark,yjyeom}@ensec.re.kr

Abstract. In this paper, we present some improved impossible differential attacks on large-block Rijndael whose block sizes are larger than 128 bits. First of all, we present some important observations which help us to significantly improve the impossible differential attacks on large-block Rijndael proposed by Nakahara-Pavão (ISC 2007). Then we introduce some new impossible differentials for large-block Rijndael. Utilizing these longer impossible differential distinguishers, together with the technique of changing the order of `MixColumns` and `AddRoundKey` operations proposed by Zhang-Wu-Feng (ICISC 2007), we can apply impossible differential attacks up to 7-round Rijndael-160, 8-round Rijndael-192, and 9-round Rijndael-224/256. As far as we know, except the attack on Rijndael-256, all the other results are the best cryptanalytic results on large-block Rijndael.

Keywords: Rijndael, Block cipher, Impossible differential distinguisher, Impossible differential attack.

1 Introduction

The block cipher Rijndael [6,7] was submitted to the NIST Advanced Encryption Standard (AES) competition, and the 128-bit block version of Rijndael was later selected as the winner and then published as FIPS-197: AES [1]. In the original design of Rijndael, both block and key size can independently range from 128 bits to 256 bits in steps of 32 bits. Therefore, there are five block size variants of Rijndael, which can be denoted as Rijndael-128/160/192/224/256, respectively. Rijndael-128 is usually known as AES, and the other variants with block size larger than 128 bits are usually called large-block Rijndael. Although the key size can also range from 128 bits to 256 bits, in this paper we only consider the case that the key size is equal to the block size for simplicity.

Although there are lots of cryptanalytic results on AES [2,3,4,8,9,11,18,23,24,25], the large-block Rijndael have not received enough attention, and there

T.-C. Wu et al. (Eds.): ISC 2008, LNCS 5222, pp. 298–315, 2008.

are few cryptanalytic results on them. Except the multiset and integral attacks on Rijndael in [19,10], the only known results on large-block Rijndael are the impossible differential attacks described in [20]. Furthermore, recent constructions of hash functions [16] and MACs [13] sometimes adopt the design strategy of Rijndael and use modified round structures with large block size as building blocks. Since the inherent properties of building blocks can endanger the security of full construction [21], it is necessary to examine the security of these large-block variants precisely.

In this paper, we first describe some important observations for analyzing Rijndael. Then combining these observations with the early abort technique [17,23], we can improve the impossible differential attacks on large-block Rijndael described in [20] by reducing both data and time complexity significantly. Then we present some new impossible differentials for large-block Rijndael. Based on these longer ID (impossible differential) distinguishers, together with the trick of changing the order of `MixColumns` and `AddRoundKey` operations used in [23], we can apply impossible differential attacks to more rounds of Rijndael.

This paper is organized as follows. Section 2 gives a brief description of Rijndael and then introduces the notations used throughout the paper. Section 3 first describes some important observations which are used to improve the impossible differential attack on Rijndael-160, and then presents a new impossible differential attack on Rijndael-160 which is based on a new longer ID distinguisher. Similarly, Section 4, Section 5 and Section 6 describe both our improved and new impossible differential attacks on Rijndael-192, Rijndael-224 and Rijndael-256, respectively. Finally, Section 7 concludes the paper.

2 Preliminaries

2.1 Description of Rijndael

Here, we give a brief description of Rijndael. For a more detailed specification of the algorithm, you can refer to [6,7].

Rijndael is a Substitute Permutation Network (SPN) block cipher. The plaintext, ciphertext and all the intermediate data are represented by a $4 \times N_b$ rectangular array of bytes, which is called the state. N_b is the number of 32-bit words in a block, namely $N_b = b/32$, where b is the block size. The byte indexing of the state is shown in Fig. 1.

The round function of Rijndael is composed of four operations: `SubBytes`(`SB`), `ShiftRows`(`SR`), `MixColumns`(`MC`) and `AddRoundKey`(`ARK`). In `SubBytes`, a single 8×8 S-box is applied to each byte of the state. `ShiftRows` is a byte transposition which cyclically shifts the rows of the state over different offsets to the left. For each block size of Rijndael, the shift offsets are listed in Fig. 2. Specifically, Row i is cyclically shifted over C_i bytes to the left, where $0 \leq i \leq 3$. The `MixColumns` operation is an MDS matrix multiplication over GF(2^8) which confuses the four bytes of each state column. Note that the branch number of `MixColumns` is 5.

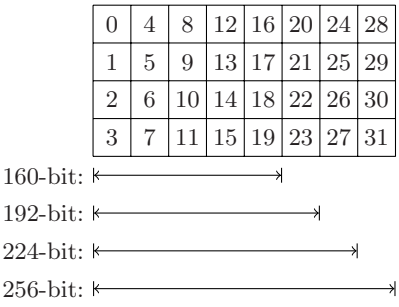
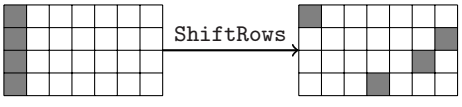


Fig. 1. Byte indexing of the state



N_b	C_0	C_1	C_2	C_3
5	0	1	2	3
6	0	1	2	3
7	0	1	2	4
8	0	1	3	4

Fig. 2. `ShiftRows` operation and the shift offsets for each block length N_b

The `AddRoundKey` is a byte XORed operation where each byte of the round subkey is XORed to the corresponding byte of the state.

In the encryption procedure of Rijndael, an additional `AddRoundKey` operation is performed before the first round, and the `MixColumns` operation is omitted in the last round. We also assume this is the case in the reduced round variants of Rijndael. Furthermore, since we have not exploited any relations between the round subkeys, we will omit the key scheduling algorithm here and interested readers can refer to [6,7].

2.2 Notations

In the following, we introduce some notations used throughout this paper. The plaintext and ciphertext are denoted as P and C respectively. We denote the XOR (bit-wise exclusive OR) operation by $\oplus$. Then the state of i-th round and its difference are denoted as X_i and ΔX_i, respectively. Let $X_{i,j}$ denote the j-th byte of X_i, and $X_{i,\,\mathrm{col}(l)}$ denote the l-th column of X_i. Let RK_i denote the subkey of the i-th round. Moreover, the input of the i-th round is denoted as X_i^I, and the intermediate values after the application of `SubBytes`, `ShiftRows`, `MixColumns` and `AddRoundKey` are denoted as X_i^S, X_i^R, X_i^M and X_i^O, respectively.

In some cases, the order of the `MixColumns` and `AddRoundKey` operations in the same round can be interchanged, which is done by replacing the subkey RK_i by an equivalent subkey RK_i^*, where $RK_i^* = MC^{-1}(RK_i)$. We use X_i^W to denote the intermediate value after the application of `AddRoundKey` operation with RK_i^* in the i-th round.

3 Impossible Differential Attacks on Rijndael-160

In this section we give two kinds of impossible differential attack on Rijndael-160.

Firstly, we improve the impossible differential attack on 6-round Rijndael-160 described in [20]. Although we use the same 4-round ID distinguisher, we can significantly reduce both data and time complexity by utilizing some important observations. Furthermore, instead of guessing all the required subkey bits simultaneously as in [20], we use the early abort technique [17,23] which guesses only a small fraction of the subkey bits to decrypt and check one column of the state each time. Then some wrong pairs can be discarded before the next guess and this can reduce lots of computation workloads.

Secondly, we present a new 5-round impossible differential for Rijndael-160. By setting this 5-round ID distinguisher in the middle rounds and analyzing one additional round both before and after the distinguisher, we can extend the impossible differential attack on Rijndael-160 up to 7-round.

3.1 Improved Impossible Differential Attack on Rijndael-160

The attack procedure of our improved impossible differential attack on 6-round Rijndael-160 is illustrated in Fig. 3. Here, the gray cell means that we have a non-zero difference for this byte, the light cell stands for zero difference, and the cell filled by the mark '?' stands for an arbitrary value.

Similar to the attack in [20], we also set the same 4-round impossible differential in the middle rounds, and recover subkey bits of RK_0 and RK_6. However, based on the following observations, we can reduce both the number of plaintext pairs needed and subkey bits guessed.

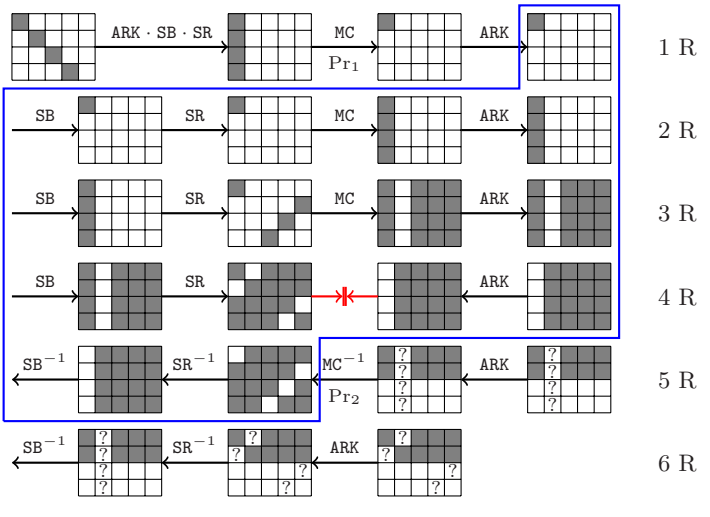


Fig. 3. Improved impossible differential attack on 6-round Rijndael-160

Observations. According to the 4-round impossible differential, if the four bytes (0,11,14,17) of ΔX_5^R are zero, then contradiction can be caused before and after the `MixColumns` operation in Round 4. Therefore, in order to check if a ciphertext pair satisfies the impossible differential, we only need to decrypt four columns of ΔX_5^O, and the second column of ΔX_5^O can take arbitrary values. Thus we can filter out the ciphertext pairs using the following 64-bit condition, namely zero ciphertext differences at bytes (2,3,6,7,10,11,14,19), instead of the 80-bit condition used in [20]. Furthermore, instead of guessing the topmost two rows (80-bit) of RK_6 as in [20], we only need to guess 64-bit subkey ($RK_{6,0}$, $RK_{6,5}$, $RK_{6,8}$, $RK_{6,9}$, $RK_{6,12}$, $RK_{6,13}$, $RK_{6,16}$, $RK_{6,17}$), since only four columns need to be decrypted.

In the following, we describe the attack procedure in details.

1. A structure is a set of 2^{32} plaintexts, in which the plaintexts take all the possible 32-bit values at the 4 bytes $(0, 5, 10, 15)$, and at the remaining bytes they take certain fixed values. Choose $2^{61.2}$ such structures, which can generate about $2^{61.2} \cdot (2^{32})^2/2 = 2^{124.2}$ plaintext pairs. Choose the pairs whose ciphertext differences at the 8 bytes $(2, 3, 6, 7, 10, 11, 14, 19)$ are zero, and the expected number of remaining pairs is $2^{124.2} \cdot 2^{-64} = 2^{60.2}$.
2. For all the remaining pairs, guess the 16-bit subkey $(RK_{6,0}, RK_{6,17})$ and partially decrypt Round 6 to compute the first column of ΔX_5^R, namely $\Delta X^R_{5,\,\mathrm{col}(0)} = MC^{-1}(ARK(\Delta X^O_{5,\,\mathrm{col}(0)}))$. Check if the first byte of this column equals to 0, and if this is not the case, discard the pair. After this test, there remains about $2^{60.2} \cdot 2^{-8} = 2^{52.2}$ pairs.
3. For every guess of the 16-bit subkey $(RK_{6,5}, RK_{6,8})$, partially decrypt Round 6 to compute the third column of ΔX_5^R. Check if the fourth byte of this column equals to 0. If not, then discard the pair. After this test, there remains about $2^{52.2} \cdot 2^{-8} = 2^{44.2}$ pairs.
4. For every guess of the 16-bit subkey $(RK_{6,9}, RK_{6,12})$, partially decrypt Round 6 to compute the fourth column of ΔX_5^R. Check if the third byte of this column equals to 0 and discard unsatisfied pairs. After this test, there remains about $2^{44.2} \cdot 2^{-8} = 2^{36.2}$ pairs.
5. For every guess of the 16-bit subkey $(RK_{6,13}, RK_{6,16})$, partially decrypt Round 6 to compute the fifth column of ΔX_5^R. Check if the second byte of this column equals to 0. Discard the unsatisfied pairs and there remains about $2^{36.2} \cdot 2^{-8} = 2^{28.2}$ pairs.
6. For all the remaining pairs, guess 32-bit subkey ($RK_{0,0}$, $RK_{0,5}$, $RK_{0,10}$, $RK_{0,15}$) to partially encrypt the first round to get the first column of ΔX_1^O. Check if there is only one nonzero byte in this column. Since such a difference is impossible, every subkey guess that leads to such a difference is wrong and thus can be deleted. After analyzing all the $2^{28.2}$ remaining pairs, if there remains value of RK_0 which is not deleted, output the 96-bit subkey guess of (RK_0, RK_6) as the correct key.

Analysis. The probability that a pair passes all the tests from Step 2 to Step 5 is about $\mathrm{Pr}_2 = (2^{-8})^4 = 2^{-32}$. The probability that a pair satisfies the condition

in Step 6 is about $\Pr_1 = 4 \cdot (2^8 - 1)/2^{32} \approx 2^{-22}$, since the `MixColumns` operation is linear and there is only one nonzero byte in one column. Therefore, after analyzing the $2^{28.2}$ remaining pairs, only about $2^{32} \cdot (1 - 2^{-22})^{2^{28.2}} \approx 2^{-74}$ wrong guess for the four bytes of RK_0 remains. Furthermore, unless the initial guess on the 64-bit subkey of RK_6 is correct, it is expected that we can get rid of all the possible 32-bit guesses for RK_0, since the wrong 96-bit subkey guess of (RK_0, RK_6) only remains with the probability of $2^{64} \cdot 2^{-74} \approx 2^{-10}$.

The data complexity of the attack is $2^{61.2} \cdot 2^{32} = 2^{93.2}$ CP(Chosen Plaintexts), and the time complexity can be estimated as follows. Step 2 requires about $2^{16} \cdot 2^{60.2} \cdot 2/5 \approx 2^{75}$ one round encryption, since in this step for each ciphertext we only decrypt one column of the state. Similarly, Step 3 requires about $2^{32} \cdot 2^{52.2} \cdot 2/5 \approx 2^{83}$ one round encryption; Step 4 and Step 5 require about 2^{91} and 2^{99} one round encryption, respectively. Step 6 requires about $2^{64} \cdot 2^{32} \cdot 2 \cdot (1 + (1 - 2^{-22}) + (1 - 2^{-22})^2 + \ldots + (1 - 2^{-22})^{2^{28.2}})/5 \approx 2^{116.7}$ one round encryption. Thus the total time complexity of the attack is about $2^{116.7}/6 \approx 2^{114.1}$ 6-round encryptions. The memory space required is 2^{96} bits to store the key guess table.

3.2 New Impossible Differential Attack on Rijndael-160

Here, we introduce a new impossible differential attack on 7-round Rijndael-160. You can easily check that a similar impossible differential pattern and attack techniques used in [23] for AES can be adopted to 7-round Rijndael-160 whose time and data complexities are 2^{119} and $2^{149.5}$, respectively. However, we can mount a better attack on 7-round Rijndael-160 by using a new 5-round impossible differential which is longer than the one used in Sect. 3.1.

The 5-round impossible differential is illustrated in Fig. 4. In the encryption and decryption direction, the differentials both start with only one nonzero byte difference. Then in Round 4, the first column of the state before and after $\mathtt{MC}^{-1}$ operation contradicts with each other, since the sum of nonzero byte differences is only 3 which is less than the branch number of `MixColumns`.

Based on this 5-round ID distinguisher, we can mount an impossible differential attack on 7-round Rijndael-160, which recovers subkey bits of RK_0 and RK_7. The attack procedure is as follows and it is also illustrated in Fig. 4.

1. Choose 2^{115} structures, and in each structure the plaintexts take all the possible 32-bit values at bytes $(0, 5, 10, 15)$, and at the remaining bytes they take certain fixed values. These structures can generate about $2^{115} \cdot (2^{32})^2/2 = 2^{178}$ plaintext pairs. Choose the pairs whose corresponding ciphertext pairs have zero differences except 4 bytes $(1, 4, 15, 18)$, and the expected number of remaining pairs is $2^{178} \cdot 2^{-128} = 2^{50}$.
2. For every guess of the 32-bit subkey ($RK_{7,1}$, $RK_{7,4}$, $RK_{7,15}$, $RK_{7,18}$), partially decrypt Round 7 to compute the second column of ΔX_6^W. Check if there is only one nonzero byte difference in this column. If this is not the case, discard the pair.
3. For all the remaining pairs, guess 32-bit subkey ($RK_{0,0}$, $RK_{0,5}$, $RK_{0,10}$, $RK_{0,15}$) to partially encrypt the first round to get the first column of ΔX_1^O.

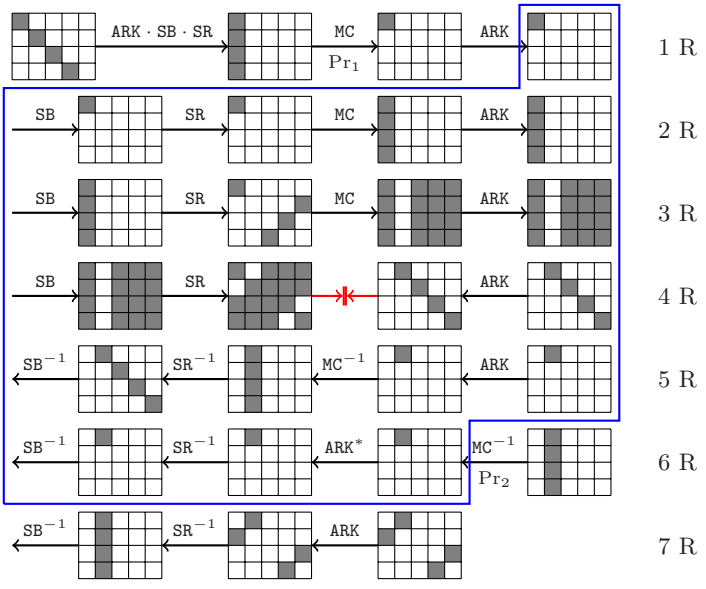


Fig. 4. New impossible differential attack on 7-round Rijndael-160

If there is only one nonzero byte in this column, the guessed subkey is wrong and should be deleted since such a difference is impossible. After analyzing all the remaining pairs, if there still remains value of RK_0 which is not deleted, output the 64-bit subkey guess of (RK_0, RK_7) as the correct key.

Analysis. The probability that a pair passes the test in Step 2 is $\mathrm{Pr}_2 = 4 \cdot (2^8 - 1)/2^{32} \approx 2^{-22}$, since the nonzero byte has 4 possible positions. Thus the expected number of remaining pairs after Step 2 is about $2^{50} \cdot 2^{-22} = 2^{28}$. Similarly, the probability which a pair satisfies the condition in Step 3 is $\mathrm{Pr}_1 \approx 2^{-22}$. Therefore, after testing all the 2^{28} remaining pairs, the probability that a wrong 64-bit subkey guess of (RK_0, RK_7) remains is about $2^{64} \cdot (1 - 2^{-22})^{2^{28}} \approx 2^{-28}$. So we can expect that only the right subkey remains.

The data complexity of the attack is $2^{115} \cdot 2^{32} = 2^{147}$ CP, and the time complexity can be estimated as follows. Step 2 requires about $2^{32} \cdot 2^{50} \cdot 2/5 \approx 2^{80.7}$ one round encryption. Step 3 requires about $2^{64} \cdot 2 \cdot (1 + (1 - 2^{-22}) + (1 - 2^{-22})^2 + \cdots + (1 - 2^{-22})^{2^{28}})/5 \approx 2^{84.7}$ one round encryption. Thus the total time complexity of the attack is about $2^{84.7}/7 \approx 2^{81.9}$ 7-round encryptions. The memory space required is about 2^{64} bits to store the key table.

4 Impossible Differential Attacks on Rijndael-192

In this section, we give three kinds of impossible differential attack on Rijndael-192. The first attack is an improved version of the impossible differential attack

on 6-round Rijndael-192 described in [20], which is based on the similar observations as the one used in Sect. 3.1. The second attack is a new impossible differential attack on 8-round Rijndael-192 which is based on the 6-round impossible differential introduced in [20, Appendix]. This 6-round impossible differential was not used in [20], because they thought it might be ineffective for a key-recovery attack due to too many zero byte differences. However, by utilizing some tricks in the filtering process, we can use this 6-round ID distinguisher to extend the impossible differential attack on Rijndael-192 up to 8-round. The third attack uses a new 6-round impossible differential which is designed by the idea that the sum of nonzero bytes of a column before and after `MixColumns` in a middle round can not be less than 5, which is the branch number of `MixColumns`. This attack can also reach up to 8-round Rijndael-192.

4.1 Improved Impossible Differential Attack on Rijndael-192

The attack procedure of our improved impossible differential attack on 6-round Rijndael-192 is illustrated in Fig. 5. Similar to the observations in Sect. 3.1, to check if a pair satisfies the impossible differential, we only need to decrypt four columns of ΔX_5^O, and the other two columns can take arbitrary values. Therefore, we only need to choose pairs whose ciphertext differences at the 8 bytes $(3, 6, 7, 10, 11, 14, 15, 18)$ are zero, which means a 64-bit condition. Then to partially decrypt four columns of ΔX_5^O, we only need to guess 64-bit subkey $(RK_{6,0}, RK_{6,9}, RK_{6,12}, RK_{6,13}, RK_{6,16}, RK_{6,17}, RK_{6,20}, RK_{6,21})$.

Since the filtering condition and the number of subkey bits guessed in this attack are the same as the one in Sect. 3.1, the data and time complexity of these two attacks should be the same. Thus the data complexity of the attack is $2^{93.2}$ CP and the time complexity is about $2^{114.1} \cdot 5/6 \approx 2^{113.8}$ 6-round encryptions.

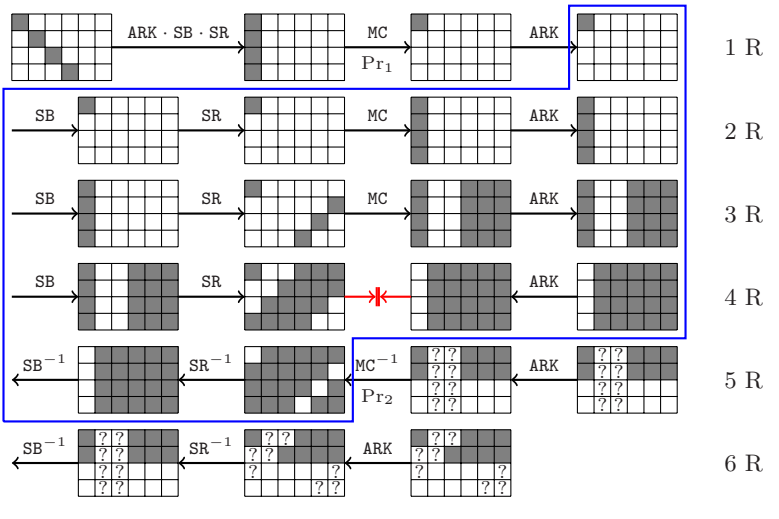


Fig. 5. Improved impossible differential attack on 6-round Rijndael-192

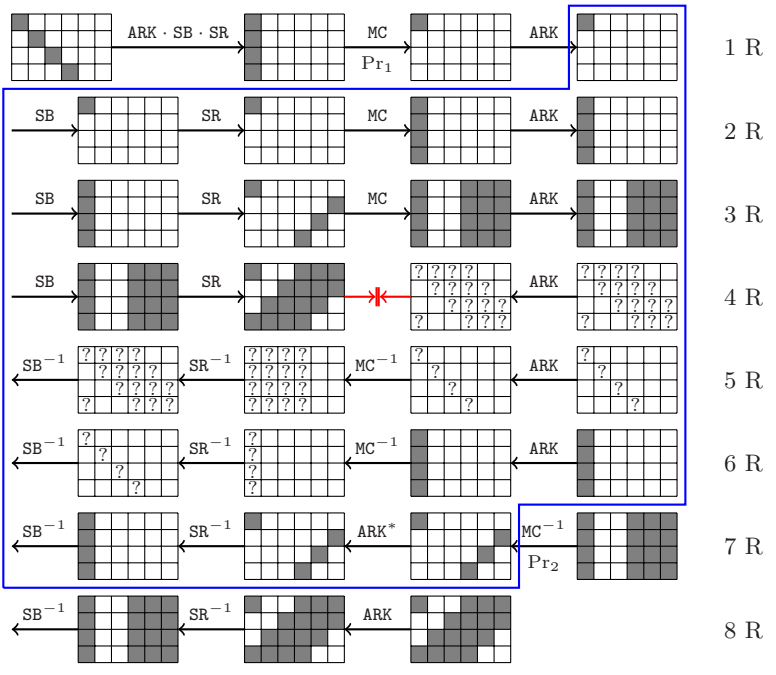


Fig. 6. First impossible differential attack on 8-round Rijndael-192

Note that in the attack on Rijndael-192, the computation of one column equals to about 1/6 one-round encryption.

4.2 New Impossible Differential Attacks on Rijndael-192

In this section, we introduce two new impossible differential attacks on 8-round Rijndael-192. Each attack uses a different 6-round impossible differential.

In the following, we describe our first impossible differential attack on 8-round Rijndael-192 which is based on the 6-round impossible differential presented in [20, Appendix]. The main idea of this ID distinguisher which is illustrated in Fig. 6 is the sum of nonzero bytes of the first column before and after the `MixColumns` operation in Round 4 is at most 4, which contradicts the branch number of `MixColumns`.

As depicted in Fig. 6, we set the above ID distinguisher in the middle rounds, and try to recover subkey bits of (RK_0, RK_8). In this attack we exploit the trick of changing the order of `MixColumns` and `AddRoundKey` operations in Round 7 to reduce the number of subkey bits guessed. This is done by replacing the subkey RK_7 with the equivalent subkey RK_7^*. Details of our attack procedure and complexity analysis are described as follows.

1. Choose 2^{126} structures which contains 2^{32} plaintexts each. The plaintexts in a structure take all the possible 32-bit values at the four bytes $(0, 5, 10, 15)$,

and at the remaining bytes they take certain fixed values. These structures can generate about $2^{126} \cdot (2^{32})^2/2 = 2^{189}$ plaintext pairs. Choose the pairs whose ciphertext differences at the 8 bytes $(1, 2, 4, 5, 8, 19, 22, 23)$ are zero, and the expected number of remaining pairs is $2^{189} \cdot 2^{-64} = 2^{125}$.

2. For all the remaining pairs, guess 32-bit subkey ($RK_{8,0}$, $RK_{8,15}$, $RK_{8,18}$, $RK_{8,21}$) and partially decrypt Round 8 to compute the first column of ΔX_7^W. Check if the differences at the 3 bytes $(1, 2, 3)$ are zero. If this is not the case, discard the pair. After this test, there remains about $2^{125} \cdot 2^{-24} = 2^{101}$ pairs.
3. For every guess of the 32-bit subkey ($RK_{8,3}$, $RK_{8,6}$, $RK_{8,9}$, $RK_{8,12}$), partially decrypt Round 8 to compute the fourth column of ΔX_7^W. Check if the differences at the 3 bytes $(12, 13, 14)$ are zero. If not, then discard the pair. The expected number of remaining pairs is about $2^{101} \cdot 2^{-24} = 2^{77}$.
4. For every guess of the 32-bit subkey ($RK_{8,7}$, $RK_{8,10}$, $RK_{8,13}$, $RK_{8,16}$), partially decrypt Round 8 to compute the fifth column of ΔX_7^W. Check if the differences at the 3 bytes $(16, 17, 19)$ are zero. If not, then discard the pair. The expected number of remaining pairs is about $2^{77} \cdot 2^{-24} = 2^{53}$.
5. For every guess of the 32-bit subkey ($RK_{8,11}$, $RK_{8,14}$, $RK_{8,17}$, $RK_{8,20}$), partially decrypt Round 8 to compute the sixth column of ΔX_7^W. Check if the differences at the 3 bytes $(20, 22, 23)$ are zero, and discard unsatisfied pairs. The expected number of remaining pairs is about $2^{53} \cdot 2^{-24} = 2^{29}$.
6. For all the remaining pairs, guess 32-bit subkey ($RK_{0,0}$, $RK_{0,5}$, $RK_{0,10}$, $RK_{0,15}$) to partially encrypt the first round to get the first column of ΔX_1^O. Check if there is only one nonzero byte in this column. Delete all the 32-bit subkey guesses of RK_0 which lead to such an impossible difference. After analyzing all the 2^{29} remaining pairs, if there still remains value of RK_0, output the 160-bit subkey guess of (RK_0, RK_8) as the correct key.

Analysis. The probability that a pair passes all the tests from Step 2 to Step 5 is $\Pr_2 \approx (2^{-24})^4 = 2^{-96}$, and the probability that a pair satisfies the condition in Step 6 is $\Pr_1 \approx 2^{-22}$. Therefore, after analyzing the 2^{29} remaining pairs, for all the wrong subkey guesses we can get rid of all the possible 32-bit values of RK_0, since the wrong guess of 160-bit (RK_0, RK_8) only remains with the probability $2^{128} \cdot 2^{32} \cdot (1 - 2^{-22})^{2^{29}} \approx 2^{-24}$.

The data complexity of the attack is $2^{126} \cdot 2^{32} = 2^{158}$ CP, and the time complexity can be estimated as follows. Step 2 requires about $2^{32} \cdot 2^{125} \cdot 2/6 \approx 2^{155.4}$ one round encryption. Similarly, Step 3 requires about $2^{64} \cdot 2^{101} \cdot 2/6 \approx 2^{163.4}$ one round encryption; Step 4 and Step 5 require about $2^{171.4}$ and $2^{179.4}$ one round encryption, respectively. Step 6 requires about $2^{128} \cdot 2^{32} \cdot 2 \cdot (1 + (1 - 2^{-22}) + (1 - 2^{-22})^2 + \cdots + (1 - 2^{-22})^{2^{29}})/6 \approx 2^{180.4}$ one round encryption. Thus the total time complexity of the attack is about $2^{180.4}/8 = 2^{177.4}$ 8-round encryptions. The memory space required is about 2^{160} bits to store the key table.

Next, we introduce our second impossible differential attack on 8-round Rijndael-192 which is based on a new 6-round impossible differential. The 6-round ID distinguisher and brief attack procedure are illustrated in Fig. 7. In this attack, we also change the order of `MixColumns` and `AddRoundKey` operations

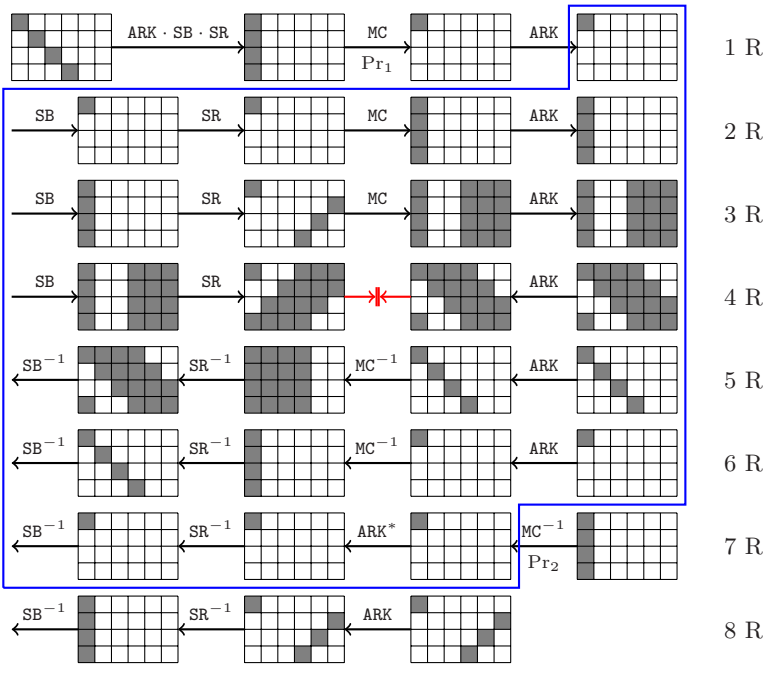


Fig. 7. Second impossible differential attack on 8-round Rijndael-192

in Round 7. Details of our attack procedure and complexity analysis are described as follows.

1. Choose 2^{147} structures, and in each structure the plaintexts take all the possible 32-bit values at bytes $(0, 5, 10, 15)$, and at the remaining bytes they take certain fixed values. These structures can generate about $2^{147} \cdot (2^{32})^2/2 = 2^{210}$ plaintext pairs. Choose only the pairs whose corresponding ciphertext pairs have zero differences except 4 bytes $(0, 15, 18, 21)$, and the expected number of remaining pairs is $2^{210} \cdot 2^{-160} = 2^{50}$.
2. For every guess of the 32-bit subkey ($RK_{8,0}$, $RK_{8,15}$, $RK_{8,18}$, $RK_{8,21}$), partially decrypt Round 8 to compute the first column of ΔX_7^W. Check if there is only one nonzero byte difference in this column. If not, then discard the pair. After this test, the expected number of remaining pairs is $2^{50} \cdot 2^{-22} = 2^{28}$.
3. For all the remaining pairs, guess 32-bit subkey ($RK_{0,0}$, $RK_{0,5}$, $RK_{0,10}$, $RK_{0,15}$) to partially encrypt the first round to get the first column of ΔX_1^O. Check if there is only one nonzero byte difference in this column. Delete all the wrong subkey guesses of RK_0 which lead to such an impossible difference. After analyzing all the remaining pairs, if there still remains value of RK_0, output the 64-bit subkey guess of (RK_0, RK_8) as the correct key.

Analysis. The probabilities that a pair passes the test in Step 2 and Step 3 are $\mathrm{Pr}_1 = \mathrm{Pr}_2 = 4 \cdot (2^8 - 1)/2^{32} \approx 2^{-22}$. Therefore, after testing the 2^{28} remaining

pairs, the probability that a wrong 64-bit subkey guess of (RK_0, RK_8) remains is $2^{64} \cdot (1 - 2^{-22})^{2^{28}} \approx 2^{-28}$. So we can expect that only the right subkey will remain. Hence he data complexity of the attack is $2^{147} \cdot 2^{32} = 2^{179}$ CP, and the time complexity is dominated by Step 3 which requires about $2^{64} \cdot 2 \cdot (1 + (1 - 2^{-22}) + (1 - 2^{-22})^2 + \cdots + (1 - 2^{-22})^{2^{28}})/6 \approx 2^{84.4}$ one round encryption. Thus the total time complexity of the attack is about $2^{84.4}/8 = 2^{81.4}$ 8-round encryptions. The memory space required is about 2^{64} bits to store the key table.

5 Impossible Differential Attacks on Rijndael-224

In this section, we also first improve the impossible differential attack on 7-round Rijndael-224 described in [20]. Then we present a new 6-round impossible differential for Rijndael-224. Based on this longer ID distinguisher, we can analyze one round before and two rounds after the distinguisher to get a new impossible differential attack on 9-round Rijndael-224.

5.1 Improved Impossible Differential Attack on Rijndael-224

Based on the similar observations, we know that only four columns of the ciphertext are sufficient to be decrypted to check if a pair satisfies the distinguisher, and the other three columns can take arbitrary values. Thus instead of the 112-bit condition and 112-bit guess of RK_7 used in [20], we only need to guess 64-bit of RK_7 and filter the ciphertext pairs using a 64-bit condition. Therefore, the data complexity of our improved attack is also $2^{93.2}$ CP, and the time complexity is about $2^{113.4}$ 7-round encryptions.

5.2 New Impossible Differential Attack on Rijndael-224

The new 6-round impossible differential used in our attack on 9-round Rijndael-224 is illustrated in Fig. 8. It is constructed by exploiting the contradiction that the sum of nonzero byte differences in the first column before and after `MixColumns` operation in Round 4 is 2, which is less than the branch number. Then, as depicted in Fig. 8, we can add one round before and two rounds after the distinguisher, and mount an impossible differential attack on 9-round Rijndael-224 which recovers subkey bits of RK_0, RK_8 and RK_9. Note in this attack, we also change the order of `MixColumns` and `AddRoundKey` operations in Round 7 and Round 8. Details of our attack procedure and complexity analysis are described as follows.

1. Choose $2^{180.3}$ structures, and the plaintexts in a structure take all the possible 32-bit values at the four bytes $(0, 5, 10, 19)$, and at the remaining bytes they take certain fixed values. These structures can generate about $2^{180.3} \cdot (2^{32})^2/2 = 2^{243.3}$ pairs. Choose only the pairs which have zero ciphertext differences at the 12 bytes (1, 2, 3, 4, 5, 8, 10, 13, 16, 19, 23, 26), and the expected number of remaining pairs is $2^{243.3} \cdot 2^{-96} = 2^{147.3}$.

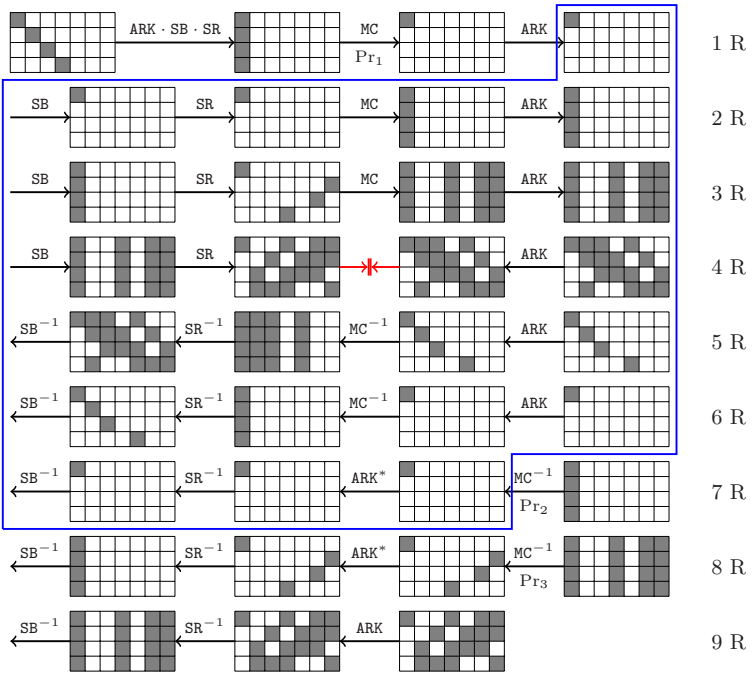


Fig. 8. New impossible differential attack on 9-round Rijndael-224

2. For all the remaining pairs, guess 32-bit subkey ($RK_{9,0}$, $RK_{9,15}$, $RK_{9,22}$, $RK_{9,25}$) and partially decrypt Round 9 to get the first column of X_8^W. Check if the difference at 3 bytes $(1, 2, 3)$ are zero. If this is not the case, discard the pair. The expected number of remaining pairs is about $2^{147.3} \cdot 2^{-24} = 2^{123.3}$.
3. For every guess of the 32-bit subkey ($RK_{9,6}$, $RK_{9,9}$, $RK_{9,12}$, $RK_{9,27}$), partially decrypt Round 9 to get the fourth column of X_8^W. Check if the difference at 3 bytes $(12, 13, 14)$ are zero. If not, then discard the pair. The expected number of remaining pairs is about $2^{123.3} \cdot 2^{-24} = 2^{99.3}$.
4. For every guess of the 32-bit subkey ($RK_{9,7}$, $RK_{9,14}$, $RK_{9,17}$, $RK_{9,20}$), partially decrypt Round 9 to get the sixth column of X_8^W. Check if the difference at 3 bytes $(20, 21, 23)$ are zero. If not, discard the pair. The expected number of remaining pairs is about $2^{99.3} \cdot 2^{-24} = 2^{75.3}$.
5. For every guess of the 32-bit subkey ($RK_{9,11}$, $RK_{9,18}$, $RK_{9,21}$, $RK_{9,24}$), partially decrypt Round 9 to get the seventh column of X_8^W. Check if the difference at 3 bytes $(24, 26, 27)$ are zero. If not, discard the pair. The expected number of remaining pairs is about $2^{75.3} \cdot 2^{-24} = 2^{51.3}$.
6. For all the remaining pairs, guess 32-bit equivalent subkey ($RK^*_{8,0}$, $RK^*_{8,15}$, $RK^*_{8,22}$, $RK^*_{8,25}$), and partially decrypt Round 8 to get the first column of ΔX_7^W. Check if there is only one nonzero byte difference. If not, discard the pair. It is expected that there remains about $2^{51.3} \cdot 4 \cdot 2^{-24} = 2^{29.3}$ pairs.

7. For the remaining pairs, guess 32-bit subkey $(RK_{0,0}, RK_{0,5}, RK_{0,10}, RK_{0,19})$ to partially encrypt the first round to get the first column of ΔX_1^O. Check if there is only one nonzero byte difference in this column. Delete all the wrong 32-bit subkey guess of RK_0 which leads to such an impossible difference. After analyzing all the $2^{29.3}$ remaining pairs, if there still remains a value of RK_0, output the 192-bit subkey guess of (RK_0, RK_8, RK_9) as correct key.

Analysis. The probability that a pair passes the tests from Step 2 to Step 5 is about $\Pr_3 = (2^{-24})^4 = 2^{-96}$. The probabilities that a pair passes the tests in Step 6 and Step 7 are $\Pr_1 = \Pr_2 = 2^{-22}$. Therefore, after analyzing the $2^{29.3}$ remaining pairs, the wrong guess of 192-bit subkey (RK_0, RK_8^*, RK_9) only remains with the probability of $2^{192} \cdot (1 - 2^{-22})^{2^{29.3}} \approx 2^{-35}$.

The data complexity of the attack is $2^{180.3} \cdot 2^{32} = 2^{212.3}$ CP, and the time complexity can be estimated as follows. Step 2 requires about $2^{32} \cdot 2^{147.3} \cdot 2/7 \approx 2^{177.5}$ one round encryption. Similarly, Step 3, Step 4, Step 5 and Step 6 require about $2^{185.5}$, $2^{193.5}$, $2^{201.5}$ and $2^{209.5}$ one round encryption, respectively. Step 7 requires about $2^{160} \cdot 2^{32} \cdot 2 \cdot (1 + (1-2^{-22}) + (1-2^{-22})^2 + \cdots + (1-2^{-22})^{2^{29.3}})/7 \approx 2^{212.2}$ one round encryption. Thus the total time complexity of the attack is about $2^{212.2}/9 \approx 2^{209}$ 9-round encryptions. The memory space required is about 2^{192} bits to store the key table.

6 Impossible Differential Attack on Rijndael-256

In this section, an improved impossible differential attack on 7-round Rijndael-256 and a new impossible differential attack on 9-round Rijndael-256 using a 6-round ID distinguisher are introduced.

6.1 Improved Impossible Differential Attack on Rijndael-256

Similar to the analysis in Sect. 3.1, in order to check if a pair satisfies the ID distinguisher, we only need to decrypt four columns of the ciphertext pairs, and the other four columns can take arbitrary values. Thus instead of the 128-bit condition and 128-bit subkey guess in [20], we only need to guess 64-bit subkey guess of RK_7 and filter out the pairs with a 64-bit condition in our improved attack on 7-round Rijndael-256. Therefore, the data complexity is also $2^{93.2}$ CP, and the time complexity is about $2^{113.2}$ 7-round encryptions.

6.2 New Impossible Differential Attack on Rijndael-256

We depict the 6-round impossible differential for Rijndael-256 in Fig. 9. Then by analyzing one round before and two rounds after the 6-round ID distinguisher, we can apply the impossible differential attack to 9-round Rijndael-256 as depicted in Fig. 9.

Considering that this attack is similar to the attack on 9-round Rijndael-192 in Sect. 5.2, we only give a brief description of the attack procedure as follows. In

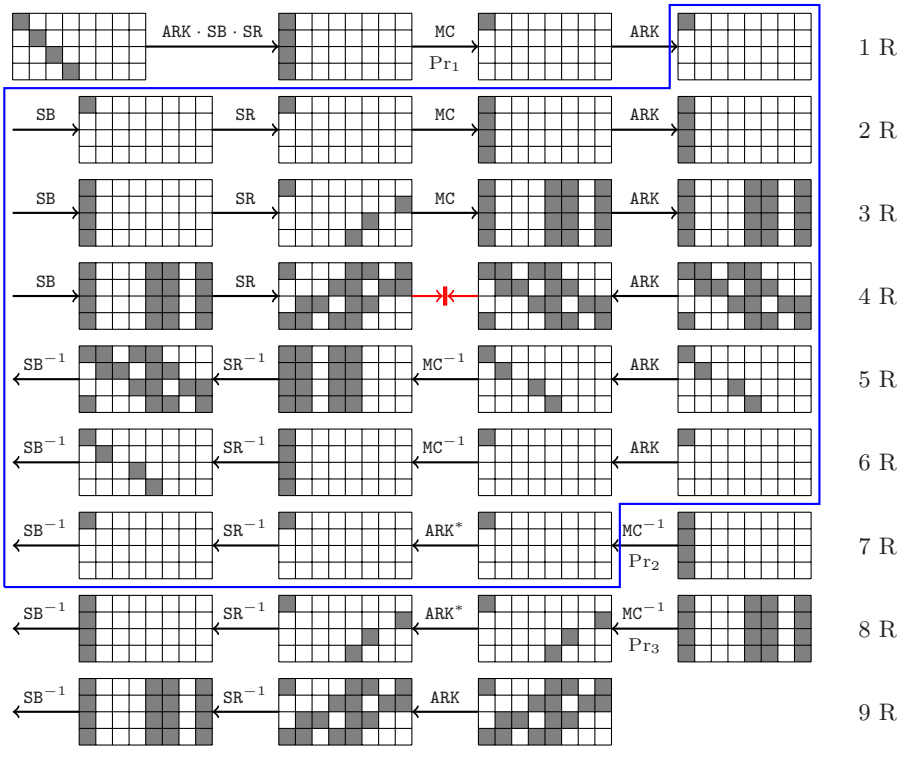


Fig. 9. New impossible differential attack on 9-round Rijndael-256

the attack, we choose $2^{212.3}$ structures which can generate about $2^{275.3}$ plaintext pairs. After filtering out the pairs based on the 128-bit condition on ciphertext differences, there remains $2^{147.3}$ pairs. Then we need to guess 128-bit of RK_9, 32-bit of RK_8^* and 32-bit of RK_0 to check if the pair satisfies the 6-round ID distinguisher. Finally, we can get rid of all the wrong 192-bit subkey guesses by analyzing the $2^{29.3}$ remaining pairs. Therefore, the data complexity of the attack is $2^{212.3} \cdot 2^{32} = 2^{244.3}$ CP, and the total time complexity of the attack is about $2^{192} \cdot 2 \cdot (1 + (1 - 2^{-22}) + (1 - 2^{-22})^2 + \cdots + (1 - 2^{-22})^{2^{29.3}})/8/9 \approx 2^{208.8}$ 9-round encryptions. The memory space required is 2^{192} bits to store the key table.

7 Conclusion

In this paper, we give two kinds of impossible differential attacks on large-block Rijndael. Firstly, we present some important observations for impossible differentials of large-block Rijndael used in [20] which induces significant improvements to the impossible differential attacks described in [20]. Next, we present some new impossible differentials for large-block Rijndael, and by using these long

ID distinguishers we extend the impossible differential attacks on large-block Rijndael to more rounds.

Table 1 summarizes our impossible differential attacks together with the previously known attacks on large-block Rijndael. According to Table 1, our improved impossible differential attack on each variant of Rijndael all makes significant improvements on both data and time complexities. Furthermore, except the attack on Rijndael-256, our new impossible differential attacks on 7-round Rijndael-160, 8-round Rijndael-192 and 9-round Rijndael-224 are all the best cryptanalytic results on large-block Rijndael so far.

Table 1. Summary of our attacks and the previously known attacks on Rijndael

Cipher	# of rounds	Complexity Time	Complexity Data(CP)	Attack type	Source
Rijndael-160	6	2^{135}	$2^{105.5}$	Imp. Diff.	[20]
	6	$2^{114.1}$	$2^{93.2}$	Imp. Diff.	Sect. 3.1
	7	$2^{133.5}$	2^{129}	Multiset	[19]
	7	$2^{81.9}$	2^{147}	Imp. Diff.	Sect. 3.2
Rijndael-192	6	2^{151}	$2^{121.5}$	Imp. Diff.	[20]
	6	$2^{113.8}$	$2^{93.2}$	Imp. Diff.	Sect. 4.1
	7	$2^{128}-2^{119}$	$2^{128}-2^{119}$	Part. Sum	[9]
	7	2^{141}	$2^{130.5}$	Multiset	[19]
	8	2^{188}	$2^{128}-2^{119}$	Part. Sum	[9]
	8	$2^{177.4}$	2^{158}	Imp. Diff.	Sect. 4.2
	8	$2^{81.4}$	2^{179}	Imp. Diff.	Sect. 4.2
Rijndael-224	7	2^{141}	$2^{130.5}$	Multiset	[19]
	7	2^{167}	2^{138}	Imp. Diff.	[20]
	7	$2^{113.4}$	$2^{93.2}$	Imp. Diff.	Sect. 5.1
	9	2^{209}	$2^{212.3}$	Imp. Diff.	Sect. 5.2
Rijndael-256	7	$2^{128}-2^{119}$	$2^{128}-2^{119}$	Part. Sum	[9]
	7	2^{141}	$2^{130.5}$	Multiset	[19]
	7	2^{44}	6×2^{32}	Integral	[10]
	7	2^{182}	2^{153}	Imp. Diff.	[20]
	7	$2^{113.2}$	$2^{93.2}$	Imp. Diff.	Sect. 6.1
	8	$2^{128}-2^{119}$	$2^{128}-2^{119}$	Integral	[10]
	9	2^{204}	$2^{128}-2^{119}$	Integral	[10]
	9	$2^{208.8}$	$2^{244.3}$	Imp. Diff.	Sect. 6.2

Acknowledgement

We would like to thank Prof. Vincent Rijmen and anonymous reviewers for helpful comments. This work is supported in part by the National High-Tech Research and Development 863 Plan of China (No.2007AA01Z470), the National

Natural Science Foundation of China (No.90604036), and the National Grand Fundamental Research 973 Program of China (No.2004CB318004).

References

1. National Institute of Standards and Technology. FIPS-197: Advanced Encryption Standard (AES) (November 2001)
2. Biham, E., Dunkelman, O., Keller, N.: Related-key impossible differential attacks on 8-round AES-192. In: Pointcheval, D. (ed.) CT-RSA 2006. LNCS, vol. 3860, pp. 21–33. Springer, Heidelberg (2006)
3. Biham, E., Keller, N.: Cryptanalysis of reduced variants of Rijadeal. Official public comment for Round 2 of the AES development effort (2000)
4. Biryukov, A.: The boomerang attack on 5 and 6 round reduced AES. In: Dobbertin, H., Rijmen, V., Sowa, A. (eds.) AES 2005. LNCS, vol. 3373, pp. 11–15. Springer, Heidelberg (2005)
5. Cheon, J.H., Kim, M.J., Kim, K., Lee, J.-Y., Kang, S.W.: Improved impossible differential cryptanlaysis of Rijndael and Crypton. In: Kim, K.-c. (ed.) ICISC 2001. LNCS, vol. 2288, pp. 39–49. Springer, Heidelberg (2002)
6. Daemen, J., Rijmen, V.: AES Proposal: Rijndael. In: 1st AES Conference, California, USA (1998)
7. Daemen, J., Rijmen, V.: The Design of Rijndael. Springer, Heidelberg (2001)
8. Demirci, H., Selçuk, A.A.: A meet-in-the-middle attack on 8-round AES. In: Nyberg, K. (ed.) FSE 2008. LNCS, vol. 5086, pp. 116–126. Springer, Heidelberg (2008)
9. Ferguson, N., Kelsey, J., Lucks, S., Schneier, B., Stay, M., Wager, D., Whiting, D.: Improved cryptanalysis of Rijndael. In: Schneier, B. (ed.) FSE 2000. LNCS, vol. 1978, pp. 213–230. Springer, Heidelberg (2001)
10. Galice, S., Minier, M.: Improving integral attacks against Rijndael-256 up to 9 rounds. In: Vaudenay, S. (ed.) AFRICACRYPT 2008. LNCS, vol. 5023, pp. 1–15. Springer, Heidelberg (2008)
11. Gilbert, H., Minier, M.: A collision attack on 7 rounds of Rijndael. In: Proc. of 3rd AES Candidate Conference (2000)
12. Jakimoski, G., Desmedt, Y.: Related-key differential cryptanalysis of 192-bit key AES variants. In: Matsui, M., Zuccherato, R. (eds.) SAC 2003. LNCS, vol. 3006, pp. 208–221. Springer, Heidelberg (2004)
13. Jakimoski, G., Subbalakshmi, K.P.: On efficient message authentication via block cipher design techniques. In: Kurosawa, K. (ed.) ASIACRYPT 2007. LNCS, vol. 4833, pp. 232–248. Springer, Heidelberg (2007)
14. Kim, J., Hong, S., Preneel, B.: Related-key rectangle attacks on reduced AES-192 and AES-256. In: Biryukov, A. (ed.) FSE 2007. LNCS, vol. 4593, pp. 225–241. Springer, Heidelberg (2007)
15. Kim, J., Hong, S., Sung, J., Lee, C., Lee, S.: Impossible differential cryptanalysis for block cipher structures. In: Johansson, T., Maitra, S. (eds.) INDOCRYPT 2003. LNCS, vol. 2904, pp. 82–96. Springer, Heidelberg (2003)
16. Knudsen, L.R., Rechberger, C., Thomsen, S.S.: The Grindahl hash functions. In: Biryukov, A. (ed.) FSE 2007. LNCS, vol. 4593, pp. 39–57. Springer, Heidelberg (2007)
17. Lu, J., Kim, J., Keller, N., Dunkelman, O.: Improving the efficiency of impossible differential cryptanalysis of reduced Camellia and MISTY1. In: Malkin, T. (ed.) CT-RSA 2008. LNCS, vol. 4964, pp. 370–386. Springer, Heidelberg (2008)

18. Lucks, S.: Attacking seven rounds of Rijndael under 192-bit and 256-bit keys. In: Proc. of 3rd AES Candidate Conference (2000)
19. Nakahara Jr., J., de Freitas, D.S., Phan, R.C.-W.: New multiset attacks on Rijndael with large blocks. In: Dawson, E., Vaudenay, S. (eds.) Mycrypt 2005. LNCS, vol. 3715, pp. 277–295. Springer, Heidelberg (2005)
20. Nakahara Jr., J., Pavão, I.C.: Impossible-differential attacks on large-block Rijndael. In: Garay, J.A., Lenstra, A.K., Mambo, M., Peralta, R. (eds.) ISC 2007. LNCS, vol. 4779, pp. 104–117. Springer, Heidelberg (2007)
21. Peyrin, T.: Cryptanalysis of GRINDAHL. In: Kurosawa, K. (ed.) ASIACRYPT 2007. LNCS, vol. 4833, pp. 551–567. Springer, Heidelberg (2007)
22. Phan, R.C.-W.: Impossible differential cryptanalysis of 7 round Advanced Encryption Standard (AES). Information Processing Letters 91, 33–38 (2004)
23. Zhang, W., Wu, W., Feng, D.: New result on impossible differential cryptanalysis of reduced AES. In: Nam, K.-H., Rhee, G. (eds.) ICISC 2007. LNCS, vol. 4817, pp. 239–250. Springer, Heidelberg (2007)
24. Zhang, W., Wu, W., Zhang, L., Feng, D.: Improved related-key impossible differential attacks on reduced-round AES-192. In: Biham, E., Youssef, A.M. (eds.) SAC 2006. LNCS, vol. 4356, pp. 15–27. Springer, Heidelberg (2007)
25. Zhang, W., Zhang, L., Wu, W., Feng, D.: Related-key differential-linear attacks on reduced AES-192. In: Srinathan, K., Pandu Rangan, C., Yung, M. (eds.) INDOCRYPT 2007. LNCS, vol. 4859, pp. 73–85. Springer, Heidelberg (2007)

A Five-Round Algebraic Property of the Advanced Encryption Standard

Jianyong Huang, Jennifer Seberry, and Willy Susilo

Centre for Computer and Information Security Research (CCISR)
School of Computer Science and Software Engineering
University of Wollongong, Australia
{jyh33,jennie,wsusilo}@uow.edu.au

Abstract. This paper presents a five-round algebraic property of the Advanced Encryption Standard (AES). In the proposed property, we modify twenty bytes from five intermediate values at some fixed locations in five consecutive rounds, and we show that after five rounds of operations, such modifications do not change the intermediate result and finally still produce the same ciphertext. We introduce an algorithm named δ, and the algorithm accepts a plaintext and a key as two inputs and outputs twenty bytes, which are used in the five-round property. We demonstrate that the δ algorithm has 20 variants for AES-128, 28 variants for AES-192 and 36 variants for AES-256. By employing the δ algorithm, we define a modified version of the AES algorithm, the δAES. The δAES calls the δ algorithm to generate twenty bytes, and uses these twenty bytes to modify the AES round keys. The δAES employs the same key scheduling algorithm, constants and round function as the AES. For a plaintext and a key, the AES and the δAES produce the same ciphertext.

Keywords: AES, A Five-Round Algebraic Property of the AES, Algorithm δ, Variants of Algorithm δ, Linear Equations, δAES.

1 Introduction

The block cipher Rijndael [1] was selected as the Advanced Encryption Standard (AES) by National Institute of Standards and Technology. Rijndael has a simple and elegant structure, and it was designed carefully to withstand two well-known cryptanalytic attacks: differential cryptanalysis [2] and linear cryptanalysis [3]. Most operations of Rijndael are based on the algebraic Galois field $GF(2^8)$, which can be implemented efficiently in dedicated hardware and in software on a wide range of processors.

Since Rijndael was adopted as a standard [4], there have been many research efforts aiming to evaluate the security of this cipher. A block cipher, named Big Encryption System (BES), was defined in [5], and Rijndael can be embedded into BES. The eXtended Linearization (XL) [6] and the eXtended Sparse Linearization (XSL) [7] techniques are new methods to solve nonlinear algebraic equations. The concept of dual ciphers was introduced in [8], and a collision attack on 7 rounds of Rijndael was proposed in [9]. The most effective attacks on

T.-C. Wu et al. (Eds.): ISC 2008, LNCS 5222, pp. 316–330, 2008.

reduced-round variants of the AES are Square attack which was used to attack the cipher Square [10,1]. The idea of the Square attack was later employed to improve the cryptanalysis of Rijndael [11], and to attack seven rounds of Rijndael under 192-bit and 256-bit keys [12]. A multiplicative masking method of AES was proposed in [13] and further discussed in [14]. The design of an AES-based stream cipher LEX was described in [15], and the construction of an AES-based message authentication code can be found in [16]. So far, no short-cut attack against the full-round AES has been found.

In this paper, we present a five round property of the AES. We modify twenty bytes from five intermediate values at some fixed locations in five consecutive rounds, and we demonstrate that after five rounds of operations, such modifications do not change the intermediate result and finally still produce the same ciphertext. We introduce an algorithm named δ, and the δ algorithm takes a plaintext and a key as two inputs and outputs twenty bytes, which are used in the five-round property. By employing the δ algorithm, we define a modified version of the AES algorithm, the δAES. The δAES calls the δ algorithm to generate twenty bytes, and uses these twenty bytes to modify the AES round keys. For a plaintext and a key, the AES and the δAES produce the same ciphertext.

This paper is organized as follows: Section 2 provides a short description of the AES. In Section 3, we present the five-round algebraic property of the AES, and introduce the δ algorithm. In Section 4, we define a modified version the AES algorithm, the δAES. Finally, Section 5 concludes this paper. Appendix A and Appendix B provide the process of finding the values of the eight variables which are used in Section 3.

2 Description of the AES

We provide a brief description of the AES, and refer the reader to [4] for a complete description of this cipher. AES is a block cipher with a 128-bit block length and supports key lengths of 128, 192 or 256 bits. For encryption, the input is a plaintext block and a key, and the output is a ciphertext block. The plaintext is first copied to 4×4 array of bytes, which is called the *state*. The bytes of a state is organized in the following format:

a_0	a_4	a_8	a_{12}
a_1	a_5	a_9	a_{13}
a_2	a_6	a_{10}	a_{14}
a_3	a_7	a_{11}	a_{15}

.

where a_i denote the i-th byte of the block. After an initial round key addition, the state array is transformed by performing a round function 10, 12, or 14 times (for 128-bit, 192-bit or 256-bit keys respectively), and the final state is the ciphertext. We denote the AES with 128-bit keys by AES-128, with 192-bit keys by AES-192, and with 256-bit keys by AES-256. Each round of AES consists of the following four transformations (the final round does not include MixColumns):

1. The SubBytes (SB) transformation. It is a non-linear byte substitution that operates independently on each byte of the state using a substitution table.
2. The ShiftRows (SR) transformation. The bytes of the state are cyclically shifted over different numbers of bytes. Row 0 is unchanged, and Row i is shifted to the left i byte cyclicly, $i \in \{1, 2, 3\}$.
3. The MixColumns (MC) transformation. It operates on the state column-by-column, considering each column as a four-term polynomial. The columns are treated as polynomials over $GF(2^8)$ and multiplied modulo $x^4 + 1$ with a fixed polynomial, written as $\{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$.
4. The AddRoundKey (ARK) transformation. A round key is added to the state by a simple bitwise exclusive-or (XOR) operation.

The key expansion of the AES generates a total of $Nb(Nr + 1)$ words: the algorithm needs an initial set of Nb words, and each of the Nr rounds requires Nb words of key data, where Nb is 4, and Nr is set to 10, 12, or 14 for 128-bit, 192-bit, or 256-bit key sizes respectively. For a 128-bit key K, we denote the round keys by

K_0^i	K_4^i	K_8^i	K_{12}^i
K_1^i	K_5^i	K_9^i	K_{13}^i
K_2^i	K_6^i	K_{10}^i	K_{14}^i
K_3^i	K_7^i	K_{11}^i	K_{15}^i

,

where i is the round number, $i \in \{1, 2, \cdots 10\}$. We note that the round key used in the initial round is the secret key K itself, and the secret key is represented without the superscript i.

3 A Five-Round Property of AES

We present a five-round property of the AES in this section. In the proposed property, we modify twenty bytes from five intermediate values at some fixed locations in five consecutive rounds, and we show that after five rounds of operations, such modifications do not change the intermediate result and finally still produce the same ciphertext. The modifications are carried out by performing four extra XOR operations at the end of each round (i.e., after the ARK transformation), and in total, we perform twenty extra XOR operations in five rounds. We require that each of these five rounds must contain SB, SR, MC and ARK transformations.

We use Figure 1 and Figure 2 to describe this property. The layout of the twenty bytes in the five intermediate values is shown in Figure 2, and the twenty bytes are G'_0, G'_2, G'_8, G'_{10}, M'_0, M'_2, M'_8, M'_{10}, R'_0, R'_2, R'_8, R'_{10}, V'_0, V'_2, V'_8, V'_{10}, Z'_0, Z'_2, Z'_8, and Z'_{10}. In Figure 1, all intermediate values are listed when using the AES algorithm to encrypt a plaintext P under a 128-bit key K , and all bytes of the intermediate values are denoted by plain variables. Correspondingly, Figure 2 enumerates all intermediate values of the AES with 20 extra XOR operations. The twenty-byte modifications take place in Round 1, 2, 3, 4 and 5, and after

ARK transformation in each of these five rounds, we perform XOR operations on Bytes 0, 2, 8 and 10. We show that the twenty-byte modifications do not change the input to Round 6, i.e., both the AES and the AES with 20 extra XOR operations generate the same input to Round 6. In Figure 2, a variable marked by a asterisk indicates that the value at that location has been affected by the twenty-byte modifications, and a plain variable shows that the value at that location is not affected by the twenty-byte modifications. For example, after ARK in Round 1 in Figure 2, Byte G_i is XORed with Byte G'_i, and after SB, we have four modified bytes H^*_i, $i \in \{0, 2, 8, 10\}$, and twelve unchanged bytes: H_1, H_3, H_4, H_5, H_6, H_7, H_9, H_{11}, H_{12}, H_{13}, H_{14}, and H_{15}.

3.1 The δ Algorithm

To decide the values of the twenty bytes: G'_i, M'_i, R'_i, V'_i and Z'_i, $i \in \{0, 2, 8, 10\}$, we introduce an algorithm named δ. For any plaintext P and any key K used in the AES algorithm, the δ algorithm accepts P and K as two inputs, and generates an output which contains twenty bytes $\{G'_i, M'_i, R'_i, V'_i, Z'_i\}$, where G'_i, M'_i, R'_i, V'_i, and Z'_i are bytes, $i \in \{0, 2, 8, 10\}$.

The δ algorithm includes a number of steps:

1. Process the first five rounds of the AES algorithm by taking the plaintext P and the key K as the inputs, i.e., start with the initial round, and process Round 1, 2, 3, 4 and 5 of the AES. Therefore, we know all intermediate values in Figure 1, from initial round to Round 5.
2. Initialize G'_i, M'_i, R'_i, V'_i and Z'_i to zero, $i \in \{0, 2, 8, 10\}$.
3. Choose G'_0, G'_2, G'_8 and G'_{10} freely. The only requirement is that at least one of these four bytes is not equal to zero, namely, G'_0, G'_2, G'_8 and G'_{10} cannot be all zeros. If G'_0, G'_2, G'_8 and G'_{10} are all zeros, the δ algorithm outputs twenty zero bytes. Once G'_0, G'_2, G'_8 and G'_{10} are decided, the remaining 16 bytes will be computed by the procedures described in Section 3.1.1, Appendix A, Appendix B and Section 3.1.2.
4. Decide M'_0, M'_2, M'_8 and M'_{10}.
5. Decide R'_0, R'_2, R'_8 and R'_{10}.
6. Decide V'_0, V'_2, V'_8 and V'_{10}.
7. Decide Z'_0, Z'_2, Z'_8 and Z'_{10}.

Remark 1. There are $2^{32} - 1$ combinations of $\{G'_0, G'_2, G'_8, G'_{10}\}$ because each byte can have 2^8 possible values.

3.1.1 Deciding M'_0, M'_2, M'_8 and M'_{10}

After we have decided the values of G'_0, G'_2, G'_8 and G'_{10}, we carry out a four-round computation (of the AES with extra 12 XOR operations), called Routine Computation One, which starts with the initial round and ends with MC in Round 4 (see Figure 2). All intermediate values from the computation of this time are stored in array called Buffer One (note that Routine Computation One produces 19 intermediate values).

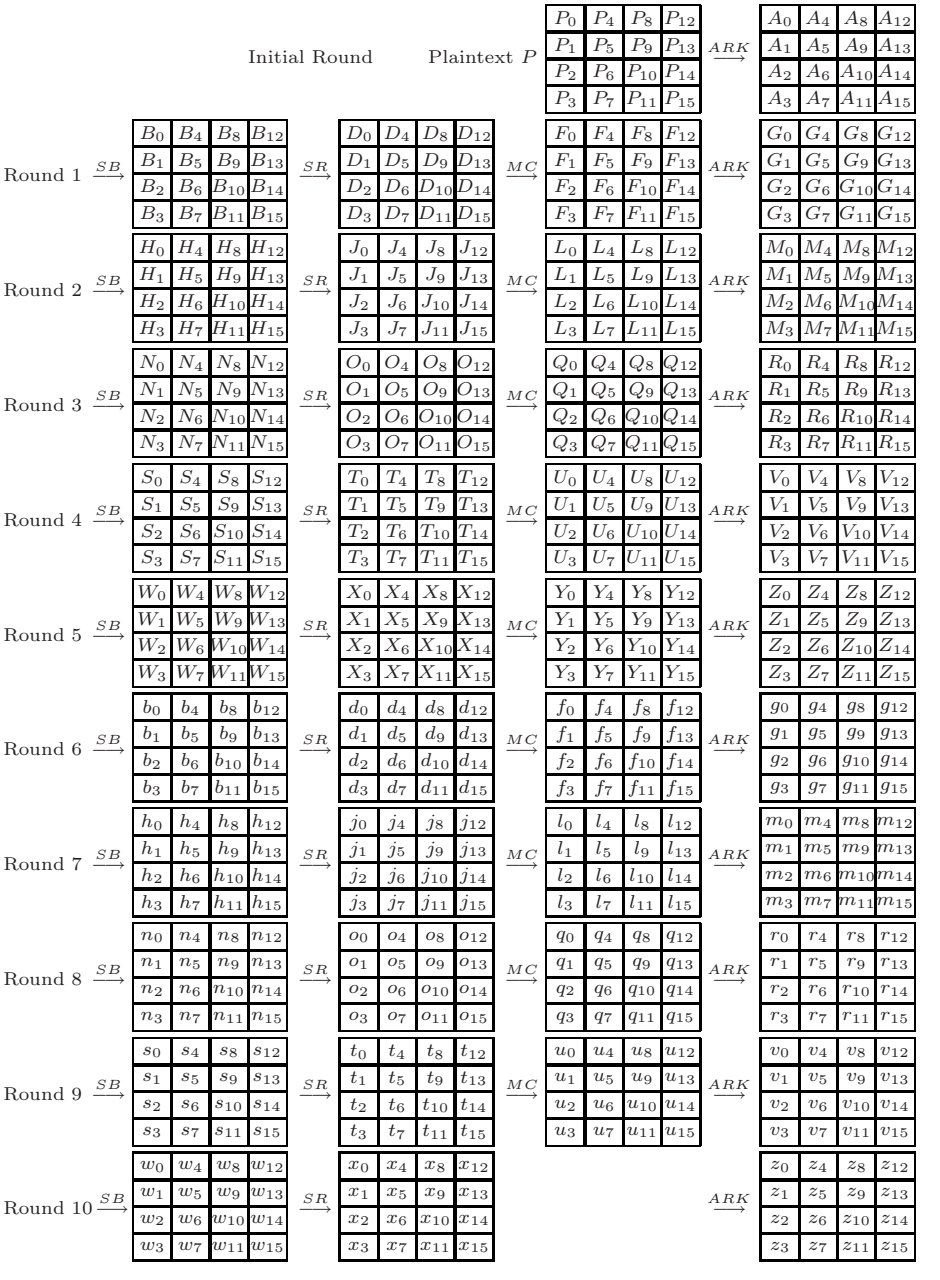


Fig. 1. The Intermediate Values of AES-128

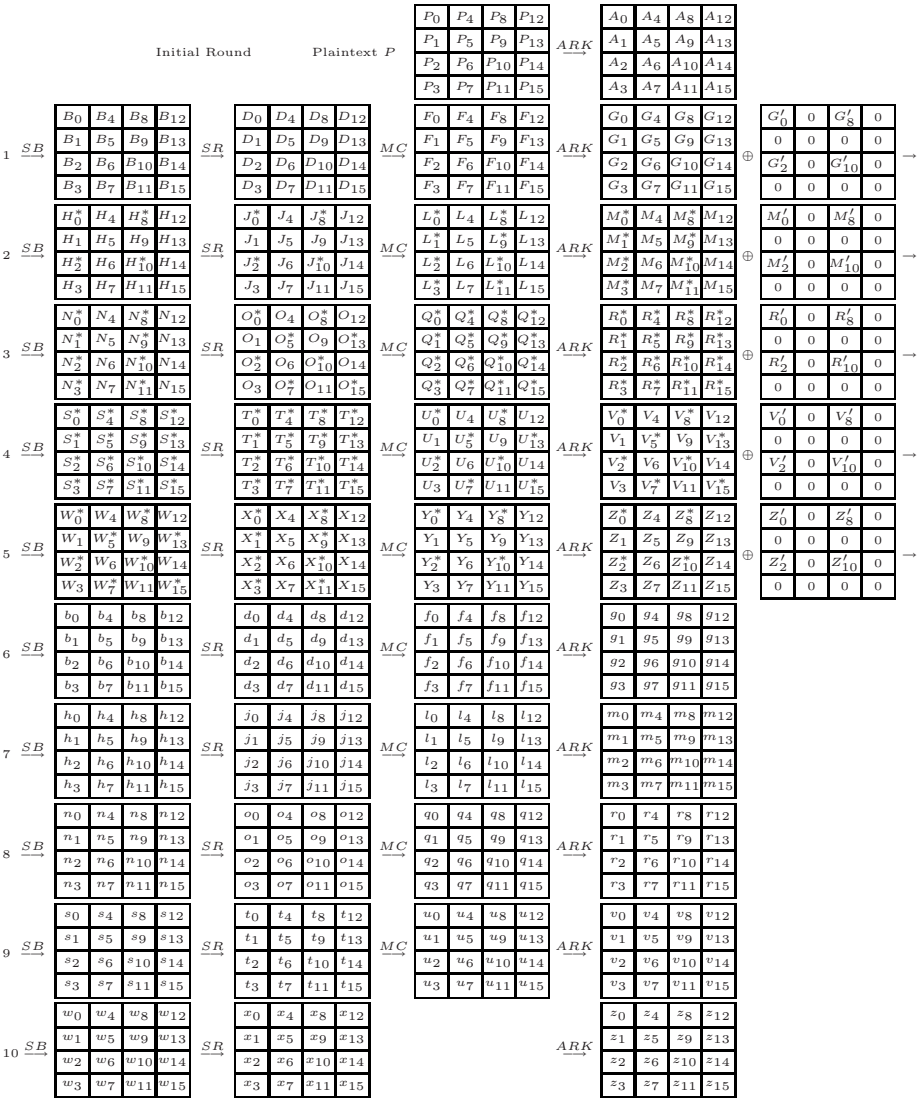


Fig. 2. The Intermediate Values of AES-128 with Extra 20 XOR Operations

Routine Computation One

$$\begin{aligned}
\texttt{Initial round}: &\xrightarrow{ARK}\\
\texttt{Round 1}: &\xrightarrow{SB}\ \xrightarrow{SR}\ \xrightarrow{MC}\ \xrightarrow{ARK}\ \xrightarrow{\oplus}\\
\texttt{Round 2}: &\xrightarrow{SB}\ \xrightarrow{SR}\ \xrightarrow{MC}\ \xrightarrow{ARK}\ \xrightarrow{\oplus}\\
\texttt{Round 3}: &\xrightarrow{SB}\ \xrightarrow{SR}\ \xrightarrow{MC}\ \xrightarrow{ARK}\ \xrightarrow{\oplus}\\
\texttt{Round 4}: &\xrightarrow{SB}\ \xrightarrow{SR}\ \xrightarrow{MC}.
\end{aligned}$$

We denote the input and output of MC in Round 4 by

$$\begin{bmatrix} T_0^* & T_4^* & T_8^* & T_{12}^* \\ T_1^* & T_5^* & T_9^* & T_{13}^* \\ T_2^* & T_6^* & T_{10}^* & T_{14}^* \\ T_3^* & T_7^* & T_{11}^* & T_{15}^* \end{bmatrix} \xrightarrow{MC} \begin{bmatrix} U_0^* & U_4^* & U_8^* & U_{12}^* \\ U_1^* & U_5^* & U_9^* & U_{13}^* \\ U_2^* & U_6^* & U_{10}^* & U_{14}^* \\ U_3^* & U_7^* & U_{11}^* & U_{15}^* \end{bmatrix}.$$

Next, we will show that there is an algebraic relation between Bytes $\{M_0', M_2', M_8', M_{10}'\}$ and Bytes $\{U_4^*, U_6^*, U_{12}^*, U_{14}^*\}$. Based on this relationship, we can change the values of $\{U_4^*, U_6^*, U_{12}^*, U_{14}^*\}$ to the values of $\{U_4, U_6, U_{12}, U_{14}\}$ by setting the values of $\{M_0', M_2', M_8', M_{10}'\}$. After we have decided the values of $\{M_0', M_2', M_8', M_{10}'\}$, we aim to have an intermediate value after MC in Round 4 in the format of

$$\begin{bmatrix} U_0^* & U_4 & U_8^* & U_{12} \\ U_1^* & U_5^* & U_9^* & U_{13}^* \\ U_2^* & U_6 & U_{10}^* & U_{14} \\ U_3^* & U_7^* & U_{11}^* & U_{15}^* \end{bmatrix}.$$

The steps of deciding $\{M_0', M_2', M_8', M_{10}'\}$ are listed as follows:

$$\{M_0', M_2', M_8', M_{10}'\} \leftarrow \{N_0^*, N_2^*, N_8^*, N_{10}^*\} \leftarrow \{O_0^*, O_2^*, O_8^*, O_{10}^*\} \leftarrow \{Q_1^*, Q_3^*, Q_9^*, Q_{11}^*\}$$

$$\leftarrow \{R_1^*, R_3^*, R_9^*, R_{11}^*\} \leftarrow \{S_1^*, S_3^*, S_9^*, S_{11}^*\} \leftarrow \{T_5^*, T_7^*, T_{13}^*, T_{15}^*\} \leftarrow \{U_4, U_6, U_{12}, U_{14}\}.$$

After we change the values of $\{U_4^*, U_6^*, U_{12}^*, U_{14}^*\}$ to the values of $\{U_4, U_6, U_{12}, U_{14}\}$, the input and output of MC in Round 4 become

$$\begin{bmatrix} T_0^* & T_4^* & T_8^* & T_{12}^* \\ T_1^* & T_5^* & T_9^* & T_{13}^* \\ T_2^* & T_6^* & T_{10}^* & T_{14}^* \\ T_3^* & T_7^* & T_{11}^* & T_{15}^* \end{bmatrix} \xrightarrow{MC} \begin{bmatrix} U_0^* & U_4 & U_8^* & U_{12} \\ U_1^* & U_5^* & U_9^* & U_{13}^* \\ U_2^* & U_6 & U_{10}^* & U_{14} \\ U_3^* & U_7^* & U_{11}^* & U_{15}^* \end{bmatrix}.$$

Our next target is to modify the values of $\{T_5^*, T_7^*, T_{13}^*, T_{15}^*\}$ according to the values of $\{U_4, U_6, U_{12}, U_{14}\}$. From the MC transformation, we have the following formula:

$$\begin{bmatrix} U_0^* & U_4 & U_8^* & U_{12} \\ U_1^* & U_5^* & U_9^* & U_{13}^* \\ U_2^* & U_6 & U_{10}^* & U_{14} \\ U_3^* & U_7^* & U_{11}^* & U_{15}^* \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} T_0^* & T_4^* & T_8^* & T_{12}^* \\ T_1^* & T_5^* & T_9^* & T_{13}^* \\ T_2^* & T_6^* & T_{10}^* & T_{14}^* \\ T_3^* & T_7^* & T_{11}^* & T_{15}^* \end{bmatrix}.$$

To find out the values of $\{T_5^*, T_7^*, T_{13}^*, T_{15}^*\}$, we need to solve the following two groups of linear functions, which are marked by (1) and (2).

$$\begin{cases} \begin{bmatrix} 02\ 03\ 01\ 01 \end{bmatrix} \begin{bmatrix} T_4^* \\ T_5^* \\ T_6^* \\ T_7^* \end{bmatrix} = U_4 \\ \begin{bmatrix} 01\ 01\ 02\ 03 \end{bmatrix} \begin{bmatrix} T_4^* \\ T_5^* \\ T_6^* \\ T_7^* \end{bmatrix} = U_6 \end{cases} \quad (1) \qquad \begin{cases} \begin{bmatrix} 02\ 03\ 01\ 01 \end{bmatrix} \begin{bmatrix} T_{12}^* \\ T_{13}^* \\ T_{14}^* \\ T_{15}^* \end{bmatrix} = U_{12} \\ \begin{bmatrix} 01\ 01\ 02\ 03 \end{bmatrix} \begin{bmatrix} T_{12}^* \\ T_{13}^* \\ T_{14}^* \\ T_{15}^* \end{bmatrix} = U_{14} \end{cases} \quad (2)$$

In (1), there are two linear equations with two undecided variables T_5^* and T_7^*, and thus we can solve (1) to obtain the values of T_5^* and T_7^*. Similarly, there are two linear equations in (2) with two undecided variables T_{13}^* and T_{15}^*, and therefore we can solve (2) to get the values of T_{13}^* and T_{15}^*. After having T_5^*, T_7^*, T_{13}^* and T_{15}^*, perform SR^{-1} (inverse ShiftRows) and SB^{-1} (inverse SubBytes), and we have the values of R_1^*, R_3^*, R_9^* and R_{11}^* after ARK in Round 3. Apply the ARK transformation to R_1^*, R_3^*, R_9^* and R_{11}^*, and we have the values of Q_1^*, Q_3^*, Q_9^* and Q_{11}^*. Our next task is to modify the values of O_0^*, O_2^*, O_8^* and O_{10}^*. In Round 3, the input and output of MC are as follows:

$$\begin{bmatrix} Q_0^* & Q_4^* & Q_8^* & Q_{12}^* \\ Q_1^* & Q_5^* & Q_9^* & Q_{13}^* \\ Q_2^* & Q_6^* & Q_{10}^* & Q_{14}^* \\ Q_3^* & Q_7^* & Q_{11}^* & Q_{15}^* \end{bmatrix} = \begin{bmatrix} 02\ 03\ 01\ 01 \\ 01\ 02\ 03\ 01 \\ 01\ 01\ 02\ 03 \\ 03\ 01\ 01\ 02 \end{bmatrix} \begin{bmatrix} O_0^* & O_4 & O_8^* & O_{12} \\ O_1 & O_5^* & O_9 & O_{13}^* \\ O_2^* & O_6 & O_{10}^* & O_{14} \\ O_3 & O_7^* & O_{11} & O_{15}^* \end{bmatrix}.$$

We can form two groups of linear equations, which are named (3) and (4), and solve them to decide O_0^*, O_2^*, O_8^* and O_{10}^*. There are two linear equations in (3) with two undetermined variables O_0^* and O_2^*, and we can solve them to determine the values of O_0^* and O_2^*. Also, there are two linear equations in (4) with two undecided variables O_8^* and O_{10}^*, and we can get O_8^* and O_{10}^* by solving (4).

$$\begin{cases} \begin{bmatrix} 01\ 02\ 03\ 01 \end{bmatrix} \begin{bmatrix} O_0^* \\ O_1 \\ O_2^* \\ O_3 \end{bmatrix} = Q_1^* \\ \begin{bmatrix} 03\ 01\ 01\ 02 \end{bmatrix} \begin{bmatrix} O_0^* \\ O_1 \\ O_2^* \\ O_3 \end{bmatrix} = Q_3^* \end{cases} \quad (3) \qquad \begin{cases} \begin{bmatrix} 01\ 02\ 03\ 01 \end{bmatrix} \begin{bmatrix} O_8^* \\ O_9 \\ O_{10}^* \\ O_{11} \end{bmatrix} = Q_9^* \\ \begin{bmatrix} 03\ 01\ 01\ 02 \end{bmatrix} \begin{bmatrix} O_8^* \\ O_9 \\ O_{10}^* \\ O_{11} \end{bmatrix} = Q_{11}^* \end{cases} \quad (4)$$

Once knowing the values of O_0^*, O_2^*, O_8^* and O_{10}^*, we perform SR^{-1} and thus we get Bytes N_0^*, N_2^*, N_8^* and N_{10}^* after SB in Round 3. Finally, Bytes M_0', M_2', M_8' and M_{10}' are decided by the following computations (note that M_0^*, M_2^*, M_8^* and M_{10}^* are obtained from Buffer One):

$$M'_0 = M^*_0 \oplus SB^{-1}(N^*_0), \quad M'_2 = M^*_2 \oplus SB^{-1}(N^*_2),$$
$$M'_8 = M^*_8 \oplus SB^{-1}(N^*_8), \quad M'_{10} = M^*_{10} \oplus SB^{-1}(N^*_{10}).$$

At this stage, we have decided the values of $\{G'_i, M'_i\}$, and $\{R'_i, V'_i, Z'_i\}$ are not yet decided (note: they are still initialized to zero), $i \in \{0, 2, 8, 10\}$.

The process of deciding R'_0, R'_2, R'_8 and R'_{10} and the routine of finding V'_0, V'_2, V'_8 and V'_{10} are similar to the steps of determining M'_0, M'_2, M'_8 and M'_{10}, and they are described in Appendix A and Appendix B respectively.

3.1.2 Deciding Z'_0, Z'_2, Z'_8 and Z'_{10}

Perform Routine Computation Two second time, and the intermediate value after MC in Round 5 is

$$\begin{bmatrix} Y^*_0 & Y_4 & Y^*_8 & Y_{12} \\ Y_1 & Y_5 & Y_9 & Y_{13} \\ Y^*_2 & Y_6 & Y^*_{10} & Y_{14} \\ Y_3 & Y_7 & Y_{11} & Y_{15} \end{bmatrix}.$$

Apply ARK to the intermediate value above, we have

$$\begin{bmatrix} Z^*_0 & Z_4 & Z^*_8 & Z_{12} \\ Z_1 & Z_5 & Z_9 & Z_{13} \\ Z^*_2 & Z_6 & Z^*_{10} & Z_{14} \\ Z_3 & Z_7 & Z_{11} & Z_{15} \end{bmatrix}.$$

Bytes Z'_0, Z'_2, Z'_8 and Z'_{10} are computed as follows: (note that Z_0, Z_2, Z_8 and Z_{10} are obtained from the computation in which the AES algorithm is used to encrypt the plaintext P under the key K (see Round 5 in Figure 1)):

$$Z'_0 = Z^*_0 \oplus Z_0, \quad Z'_2 = Z^*_2 \oplus Z_2,$$
$$Z'_8 = Z^*_8 \oplus Z_8, \quad Z'_{10} = Z^*_{10} \oplus Z10.$$

Finally, we have decided all values of $\{G'_i, M'_i, R'_i, V'_i, Z'_i\}$, $i \in \{0, 2, 8, 10\}$. Now, we carry out a five-round computation of the AES with extra 20 XOR operations, called Routine Computation Three, by using Bytes G'_0, G'_2, G'_8, G'_{10}, M'_0, M'_2, M'_8, M'_{10}, R'_0, R'_2, R'_8, R'_{10}, V'_0, V'_2, V'_8, V'_{10}, Z'_0, Z'_2, Z'_8, and Z'_{10}, and we will get the same input to Round 6 as the AES algorithm.

Routine Computation Three

$$\begin{aligned}
\texttt{Initial round}&: \xrightarrow{ARK} \\
\texttt{Round 1}&: \xrightarrow{SB} \xrightarrow{SR} \xrightarrow{MC} \xrightarrow{ARK} \xrightarrow{\oplus} \\
\texttt{Round 2}&: \xrightarrow{SB} \xrightarrow{SR} \xrightarrow{MC} \xrightarrow{ARK} \xrightarrow{\oplus} \\
\texttt{Round 3}&: \xrightarrow{SB} \xrightarrow{SR} \xrightarrow{MC} \xrightarrow{ARK} \xrightarrow{\oplus} \\
\texttt{Round 4}&: \xrightarrow{SB} \xrightarrow{SR} \xrightarrow{MC} \xrightarrow{ARK} \xrightarrow{\oplus} \\
\texttt{Round 5}&: \xrightarrow{SB} \xrightarrow{SR} \xrightarrow{MC} \xrightarrow{ARK} \xrightarrow{\oplus}.
\end{aligned}$$

Remark 2. The most important part of the δ algorithm is solving those eight groups of linear equations: (1), (2), (3), (4), (5), (6), (7) and (8). There is one question needs to be answered. The question is: are these eight groups of linear equations independent? The answer to this question is choosing different values of Bytes G'_0, G'_2, G'_8, G'_{10} if we face such situations. Among the twenty bytes: G'_0, G'_2, G'_8, G'_{10}, M'_0, M'_2, M'_8, M'_{10}, R'_0, R'_2, R'_8, R'_{10}, V'_0, V'_2, V'_8, V'_{10}, Z'_0, Z'_2, Z'_8, and Z'_{10}, we can select the values of G'_0, G'_2, G'_8 and G'_{10} freely. As we showed in Remark 1, there are $2^{32}-1$ combinations of these four bytes, and correspondingly, we can have $2^{32}-1$ intermediate values in Figure 2, starting with SB in Round 2 and ending with ARK in Round 10. If we meet any dependent equations, we can overcome this problem by choosing different values of Bytes G'_0, G'_2, G'_8 and G'_{10}. Therefore, this question will not cause any trouble. So far, we have not met any dependent equations in our large-sample experiments.

Remark 3. From Remark 1, we note that there is more than one combination of the twenty output bytes of Algorithm δ for a given pair of (P, K).

Remark 4. For distinct plaintext and cipher key pairs (P, K), Algorithm δ needs to perform individual computations to decide the values of the twenty bytes.

3.2 Variants of Algorithm δ

We show that there are other variants of the δ algorithm. In section 3.1, the locations of the twenty bytes are $\{0, 2, 8, 10\}$, and there are three other combinations, which are $\{4, 6, 12, 14\}$, $\{1, 3, 9, 11\}$ and $\{5, 7, 13, 15\}$. In Figure 2, $\{G'_i, M'_i, R'_i, V'_i, Z'_i\}$ operate in Round $\{1, 2, 3, 4, 5\}$, and they can also operate in Round $\{2, 3, 4, 5, 6\}$, $\{3, 4, 5, 6, 7\}$, $\{4, 5, 6, 7, 8\}$, or Round $\{5, 6, 7, 8, 9\}$. Therefore, there are 4 different combinations for the byte locations, and there are 5 different combinations for the round numbers in AES-128. In total, there are 20 $(= 4 \times 5)$ variants of the δ algorithm for AES-128. The δ algorithm has 28 $(= 4 \times 7)$ variants for AES-192, and 36 $(= 4 \times 9)$ variants for AES-256.

4 The Modified Version of the AES: δAES

By employing the δ algorithm, we propose a modified version of the AES, which is named δAES. The major difference between the AES and the δAES is that the δAES uses modified AES round keys. In Figure 2 in Section 3, we apply 20 extra XOR operations to the intermediate values after ARK in Round 1, 2, 3, 4 and 5 by using Bytes $\{G'_i, M'_i, R'_i, V'_i, Z'_i\}, i \in \{0, 2, 8, 10\}$. The construction of the δAES comes from the fact that we can use Bytes $\{G'_i, M'_i, R'_i, V'_i, Z'_i\}$ to XOR with AES Round Key 1, 2, 3, 4 and 5 (instead of with the intermediate values after ARK), and we still get the same result, $i \in \{0, 2, 8, 10\}$. There are twenty-byte differences between the AES round keys and the δAES round keys. The δAES employs the same key scheduling algorithm, constants and round function (i.e., SubBytes, ShiftRows, MixColumns and AddRoundKey) as the AES.

The construction of the δAES is adding two procedures, which are calling the δ algorithm and modifying the AES round keys, to the AES algorithm.

1. Suppose for a plaintext P and a cipher key K, the AES algorithm produces a ciphertext C, written as $C = AES_K(P)$.
2. By accepting P and K as two inputs, use the δ algorithm to generate twenty output bytes:

$$\{G'_i, M'_i, R'_i, V'_i, Z'_i\}, \qquad i \in \{0, 2, 8, 10\}.$$[1]

3. Apply the AES key scheduling algorithm to K and get the round keys.
4. Use $\{G'_i, M'_i, R'_i, V'_i, Z'_i\}$ to XOR with the corresponding AES round keys and get the round keys for the δAES, $i \in \{0, 2, 8, 10\}$.
5. After carrying out the transformations above, the δAES uses the same round function (i.e., SubBytes, ShiftRows, MixColumns and AddRoundKey) to process the plaintext P with modified AES round keys, and finally the δAES also generates the same ciphertext C, denoted by $C = \delta AES_K(P)$.

Compared with the AES algorithm, the δAES needs to do some extra transformations, i.e., calling the δ algorithm and modifying the AES round keys. Moreover, for distinct plaintext and cipher key pairs (P, K), the δAES needs to carry out individual computations to get Bytes $\{G'_i, M'_i, R'_i, V'_i, Z'_i\}$, $\in \{0, 2, 8, 10\}$.

5 Conclusions

We described a five-round algebraic property of the AES algorithm. In the presented property, we modify twenty bytes from five intermediate values at some fixed locations in five consecutive rounds by carrying out twenty extra XOR operations, and we show that after five rounds of processing, such modifications do not change the intermediate result and finally still produce the same ciphertext. We defined an algorithm named δ, and the δ algorithm takes a plaintext and a cipher key as two inputs and outputs twenty bytes, which are used in the five-round property. By employing the δ algorithm, we proposed a modified version of the AES algorithm, the δAES. The δAES uses the δ algorithm to generate twenty output bytes, which are used to modify the AES round keys. For a plaintext and a key, the AES and the δAES produce the same ciphertext.

References

1. Daemen, J., Rijmen, V.: AES Proposal: Rijndael, AES Round 1 Technical Evaluation CD-1: Documentation, National Institute of Standards and Technology (1998)
2. Biham, E., Shamir, A.: Differential Cryptanalysis of the Data Encryption Standard. Springer, Heidelberg (1993)
3. Matsui, M.: Linear Cryptoanalysis Method for DES Cipher. In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 386–397. Springer, Heidelberg (1994)

[1] For simplicity, we use only one variant of the δ algorithm here. Other variants of the δ algorithm also work.

4. NIST: Federal Information Processing Standards (FIPS) 197: Advanced Encryption Standard (AES). National Institute of Standards and Technology (November 26, 2001)
5. Murphy, S., Robshaw, M.: Essential Algebraic Structure within the AES. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 1–16. Springer, Heidelberg (2002)
6. Courtois, N., Klimov, A., Patarin, J., Shamir, A.: Efficient Algorithms for Solving Overdefined Systems of Multivariate Polynomial Equations. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 392–407. Springer, Heidelberg (2000)
7. Courtois, N., Pieprzyk, J.: Cryptanalysis of Block Ciphers with Overdefined Systems of Equations. In: Zheng, Y. (ed.) ASIACRYPT 2002. LNCS, vol. 2501, pp. 267–287. Springer, Heidelberg (2002)
8. Barkan, E., Biham, E.: In How Many Ways Can You Write Rijndael? In: Zheng, Y. (ed.) ASIACRYPT 2002. LNCS, vol. 2501, pp. 160–175. Springer, Heidelberg (2002)
9. Gilbert, H., Minier, M.: A Collision Attack on 7 Rounds of Rijndael. In: The Third Advanced Encryption Standard Candidate Conference, pp. 230–241 (2000)
10. Daemen, J., Knudsen, L., Rijmen, V.: The Block Cipher Square. In: Biham, E. (ed.) FSE 1997. LNCS, vol. 1267, pp. 149–165. Springer, Heidelberg (1997)
11. Ferguson, N., Kelsey, J., Lucks, S., Schneier, B., Stay, M., Wagner, D., Whiting, D.: Improved Cryptanalysis of Rijndael. In: Schneier, B. (ed.) FSE 2000. LNCS, vol. 1978, pp. 213–230. Springer, Heidelberg (2001)
12. Lucks, S.: Attacking Seven Rounds of Rijndael under 192-bit and 256-bit Keys. In: The Third Advanced Encryption Standard Candidate Conference, pp. 215–229 (2000)
13. Akkar, M.L., Giraud, C.: An Implementation of DES and AES, Secure against Some Attacks. In: Koç, Ç.K., Naccache, D., Paar, C. (eds.) CHES 2001. LNCS, vol. 2162, pp. 309–318. Springer, Heidelberg (2001)
14. Golic, J., Tymen, C.: Multiplicative Masking and Power Analysis of AES. In: Kaliski Jr., B.S., Koç, Ç.K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 198–212. Springer, Heidelberg (2003)
15. Biryukov, A.: The Design of a Stream Cipher LEX. In: Biham, E., Youssef, A.M. (eds.) SAC 2006. LNCS, vol. 4356, pp. 67–75. Springer, Heidelberg (2007)
16. Daemen, J., Rijmen, V.: A New MAC Construction ALRED and a Specific Instance ALPHA-MAC. In: Gilbert, H., Handschuh, H. (eds.) FSE 2005. LNCS, vol. 3557, pp. 1–17. Springer, Heidelberg (2005)

A Deciding R'_0, R'_2, R'_8 and R'_{10}

Perform Routine Computation One second time, and all intermediate values from the computation of this time are stored in an array called Buffer Two. The intermediate value after MC in Round 4 is

$$\begin{bmatrix} U_0^* & U_4 & U_8^* & U_{12} \\ U_1^* & U_5^* & U_9^* & U_{13}^* \\ U_2^* & U_6 & U_{10}^* & U_{14} \\ U_3^* & U_7^* & U_{11}^* & U_{15}^* \end{bmatrix}.$$

We will demonstrate that there is an algebraic relation between Bytes $\{R'_0, R'_2, R'_8, R'_{10}\}$ and Bytes $\{U_1^*, U_3^*, U_9^*, U_{11}^*\}$. By employing this relationship, we are able to change the values of $\{U_1^*, U_3^*, U_9^*, U_{11}^*\}$ to the values of $\{U_1, U_3, U_9, U_{11}\}$ by choosing the values of $\{R'_0, R'_2, R'_8, R'_{10}\}$. The moves of determining the values of $\{R'_0, R'_2, R'_8, R'_{10}\}$ are shown below:

$$\{R'_0, R'_2, R'_8, R'_{10}\} \leftarrow \{S_0^*, S_2^*, S_8^*, S_{10}^*\} \leftarrow \{T_0^*, T_2^*, T_8^*, T_{10}^*\} \leftarrow \{U_1, U_3, U_9, U_{11}\}.$$

After we replace the values of $\{U_1^*, U_3^*, U_9^*, U_{11}^*\}$ with the values of $\{U_1, U_3, U_9, U_{11}\}$, the input and output of MC in Round 4 are

$$\begin{bmatrix} T_0^* & T_4^* & T_8^* & T_{12}^* \\ T_1^* & T_5^* & T_9^* & T_{13}^* \\ T_2^* & T_6^* & T_{10}^* & T_{14}^* \\ T_3^* & T_7^* & T_{11}^* & T_{15}^* \end{bmatrix} \xrightarrow{MC} \begin{bmatrix} U_0^* & U_4 & U_8^* & U_{12} \\ U_1 & U_5^* & U_9 & U_{13}^* \\ U_2^* & U_6 & U_{10}^* & U_{14} \\ U_3 & U_7^* & U_{11} & U_{15}^* \end{bmatrix}.$$

We need to modify the values of $\{T_0^*, T_2^*, T_8^*, T_{10}^*\}$ according to the values of $\{U_1, U_3, U_9, U_{11}\}$. We can form two groups of linear equations, which are named (5) and (6). There are two undecided variables T_0^* and T_2^* in (5), and we can solve (5) to get the values of T_0^* and T_2^*. In (6), there are two undetermined variables T_8^* and T_{10}^*, and we can find out the values of T_8^* and T_{10}^* by solving (6).

$$\begin{cases} \begin{bmatrix} 01 & 02 & 03 & 01 \end{bmatrix} \begin{bmatrix} T_0^* \\ T_1^* \\ T_2^* \\ T_3^* \end{bmatrix} = U_1 \\ \begin{bmatrix} 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} T_0^* \\ T_1^* \\ T_2^* \\ T_3^* \end{bmatrix} = U_3 \end{cases} \quad (5) \qquad \begin{cases} \begin{bmatrix} 01 & 02 & 03 & 01 \end{bmatrix} \begin{bmatrix} T_8^* \\ T_9^* \\ T_{10}^* \\ T_{11}^* \end{bmatrix} = U_9 \\ \begin{bmatrix} 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} T_8^* \\ T_9^* \\ T_{10}^* \\ T_{11}^* \end{bmatrix} = U_{11} \end{cases} \quad (6)$$

After knowing the values of $\{T_0^*, T_2^*, T_8^*, T_{10}^*\}$, we perform SR^{-1} and have four corresponding values $\{S_0^*, S_2^*, S_8^*, S_{10}^*\}$ after SB in Round 4. Bytes R'_0, R'_2, R'_8

and R'_{10} are computed as follows: (note that R^*_0, R^*_2, R^*_8 and R^*_{10} are obtained from Buffer Two):

$$R'_0 = R^*_0 \oplus SB^{-1}(S^*_0), \quad R'_2 = R^*_2 \oplus SB^{-1}(S^*_2),$$

$$R'_8 = R^*_8 \oplus SB^{-1}(S^*_8), \quad R'_{10} = R^*_{10} \oplus SB^{-1}(S^*_{10}).$$

At this moment, we have decided the values of $\{G'_i, M'_i, R'_i\}$, and $\{V'_i, Z'_i\}$ are not determined and they are still equal to their initial values, $i \in \{0, 2, 8, 10\}$.

B Deciding V'_0, V'_2, V'_8 and V'_{10}

After having the values of R'_0, R'_2, R'_8 and R'_{10}, we carry out a five-round computation of the AES with 16 extra XOR operations, called Routine Computation Two, which begins with the initial round and ends with MC in Round 5 (See Figure 2). All intermediate values from the computation of this time are stored in an array named Buffer Three (note that Routine Computation Two generates 24 intermediate values).

$$\begin{aligned}
&\texttt{Routine Computation Two}\\
&\qquad\texttt{Initial round}: \xrightarrow{ARK}\\
&\qquad\texttt{Round 1}: \xrightarrow{SB} \xrightarrow{SR} \xrightarrow{MC} \xrightarrow{ARK} \xrightarrow{\oplus}\\
&\qquad\texttt{Round 2}: \xrightarrow{SB} \xrightarrow{SR} \xrightarrow{MC} \xrightarrow{ARK} \xrightarrow{\oplus}\\
&\qquad\texttt{Round 3}: \xrightarrow{SB} \xrightarrow{SR} \xrightarrow{MC} \xrightarrow{ARK} \xrightarrow{\oplus}\\
&\qquad\texttt{Round 4}: \xrightarrow{SB} \xrightarrow{SR} \xrightarrow{MC} \xrightarrow{ARK} \xrightarrow{\oplus}\\
&\qquad\texttt{Round 5}: \xrightarrow{SB} \xrightarrow{SR} \xrightarrow{MC}.
\end{aligned}$$

After MC in Round 5, we will have an intermediate value in the following format:

$$\begin{bmatrix} Y^*_0 & Y_4 & Y^*_8 & Y_{12} \\ Y^*_1 & Y_5 & Y^*_9 & Y_{13} \\ Y^*_2 & Y_6 & Y^*_{10} & Y_{14} \\ Y^*_3 & Y_7 & Y^*_{11} & Y_{15} \end{bmatrix}.$$

There is an algebraic relation between Bytes $\{V'_0, V'_2, V'_8, V'_{10}\}$ and Bytes $\{Y^*_1, Y^*_3, Y^*_9, Y^*_{11}\}$, and we can change the values of $\{Y^*_1, Y^*_3, Y^*_9, Y^*_{11}\}$ to the values of $\{Y_1, Y_3, Y_9, Y_{11}\}$ by setting the values of $\{V'_0, V'_2, V'_8, V'_{10}\}$. The steps of determining the values of $\{V'_0, V'_2, V'_8, V'_{10}\}$ are shown below:

$$\{V'_0, V'_2, V'_8, V'_{10}\} \leftarrow \{W^*_0, W^*_2, W^*_8, W^*_{10}\} \leftarrow \{X^*_0, X^*_2, X^*_8, X^*_{10}\} \leftarrow \{Y_1, Y_3, Y_9, Y_{11}\}.$$

We replace Bytes $\{Y^*_1, Y^*_3, Y^*_9, Y^*_{11}\}$ with Bytes $\{Y_1, Y_3, Y_9, Y_{11}\}$, and the input and output of MC in Round 5 are

$$\begin{bmatrix} X_0^* & X_4 & X_8^* & X_{12} \\ X_1^* & X_5 & X_9^* & X_{13} \\ X_2^* & X_6 & X_{10}^* & X_{14} \\ X_3^* & X_7 & X_{11}^* & X_{15} \end{bmatrix} \xrightarrow{MC} \begin{bmatrix} Y_0^* & Y_4 & Y_8^* & Y_{12} \\ Y_1 & Y_5 & Y_9 & Y_{13} \\ Y_2^* & Y_6 & Y_{10}^* & Y_{14} \\ Y_3 & Y_7 & Y_{11} & Y_{15} \end{bmatrix}.$$

We form two groups of linear functions, marked by (7) and (8). There are two undecided variables X_0^* and X_2^* in (7), and we can solve (7) to get the values of X_0^* and X_2^*. In (8), there are two undecided variables X_8^* and X_{10}^*, and we can obtain the values of X_8^* and X_{10}^* by solving (8).

$$\begin{cases} \begin{bmatrix} 01 & 02 & 03 & 01 \end{bmatrix} \begin{bmatrix} X_0^* \\ X_1^* \\ X_2^* \\ X_3^* \end{bmatrix} = Y_1 \\ \begin{bmatrix} 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} X_0^* \\ X_1^* \\ X_2^* \\ X_3^* \end{bmatrix} = Y_3 \end{cases} \quad (7) \qquad \begin{cases} \begin{bmatrix} 01 & 02 & 03 & 01 \end{bmatrix} \begin{bmatrix} X_8^* \\ X_9^* \\ X_{10}^* \\ X_{11}^* \end{bmatrix} = Y_9 \\ \begin{bmatrix} 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} X_8^* \\ X_9^* \\ X_{10}^* \\ X_{11}^* \end{bmatrix} = Y_{11} \end{cases} \quad (8)$$

After deciding the values of $\{X_0^*, X_2^*, X_8^*, X_{10}^*\}$, we perform SR^{-1} and have four corresponding values $\{W_0^*, W_2^*, W_8^*, W_{10}^*\}$ after SB in Round 5. Bytes V_0', V_2', V_8' and V_{10}' are computed as follows: (note that V_0^*, V_2^*, V_8^* and V_{10}^* are obtained from Buffer Three):

$$V_0' = V_0^* \oplus SB^{-1}(W_0^*), \quad V_2' = V_2^* \oplus SB^{-1}(W_2^*),$$

$$V_8' = V_8^* \oplus SB^{-1}(W_8^*), \quad V_{10}' = V_{10}^* \oplus SB^{-1}(W_{10}^*).$$

At this stage, we have decided the values of $\{G_i', M_i', R_i', V_i'\}$, and Z_i' is not determined and it is equal to the initial value, $i \in \{0, 2, 8, 10\}$.

Vortex: A New Family of One-Way Hash Functions Based on AES Rounds and Carry-Less Multiplication

Shay Gueron[1,2] and Michael E. Kounavis[3,*]

[1] Department of Mathematics, Faculty of Science and Science Education, University of Haifa, Haifa, Israel
[2] Mobility Group, Intel Corporation, Intel Design Center, Haifa, Israel
shay@math.haifa.ac.il, michael.e.kounavis@i
[3] Corporate Technology Group, Intel Corporation, Hillsboro, OR, USA

Abstract. We present Vortex a new family of one way hash functions that can produce message digests of 256 bits. The main idea behind the design of these hash functions is that we use well known algorithms that can support very fast diffusion in a small number of steps. We also balance the cryptographic strength that comes from iterating block cipher rounds with SBox substitution and diffusion (like Whirlpool) against the need to have a lightweight implementation with as small number of rounds as possible. We use only 3 AES rounds but with a stronger key schedule. Our goal is not to protect a secret symmetric key but to support perfect mixing of the bits of the input into the hash value. Three AES rounds are followed by our variant of Galois Field multiplication. This achieves cross-mixing between 128-bit sets. We present a set of qualitative arguments why we believe Vortex is secure.

1 Introduction

Guaranteeing message and code integrity is very important for the security of applications, operating systems and the network infrastructure of the future Internet. Protection against intentional alteration of data is typically supported using one way hash functions [3, 4, 5, 6, 9, 10]. A one way hash function is a mathematical construct that accepts as input a message of some length and returns a digest of much smaller length. One way hash functions are designed in such a way that it is computationally infeasible to find the input message by knowing only the digest. One way hash functions which have been in use today include algorithms like MD-5 and SHA1 [11]. One way hash functions which are likely to be used in the future include SHA256, SHA384 and SHA512 [10]. The problem with using these algorithms is that they are time consuming when implemented in software. One way hash functions typically involve multiple shifts, XOR and ADD operations which they combine in multiple rounds in order to produce message digests. Because of this reason, one way hash functions consume a substantial number of processor clocks when executing, which limits their applicability to high speed secure

* Corresponding author.

T.-C. Wu et al. (Eds.): ISC 2008, LNCS 5222, pp. 331–340, 2008.

network applications (e.g., 10 Gbps e-commerce transactions), or protection against malware (e.g., virus detection or hashed code execution).

In this document we describe an alternative approach where a family of one way hash functions is built from other security algorithms used as building blocks, which help with achieving fast mixing across a large number of input bits. Using the Merkle-Damgård construction [6, 7] as a framework we construct a compression function from AES rounds [1] and a novel merging technique based on Galois Field (GF(2)) multiplication. Using three successive AES rounds we provide mixing across 128 bits. Using a merging function based on Galois Field (GF(2)) multiplication we provide mixing across sets of 128 bits. Perfect mixing is accomplished through combinations of AES rounds and our merging function.

We have conducted 2^{20} experiments computing the collision resistance and the randomness of the output differentials of our family. Whereas our work is in progress our initial results indicate that there is no experimental evidence that Vortex is inferior when compared to SHA256. Performance-wise, however, the difference can be substantial. SHA256 operates at 21 cycles per byte on an Intel® Core 2 Duo processor. Vortex operates at a speed of 1.5 cycles per byte in a hypothetical CPU with the same micro-architecture but also with instruction set support for AES round computation and Galois Field (GF(2)) multiplication, which is the current trend in the processor industry. We believe that the design of the Vortex family is important because it represents a scalable on-the-CPU solution for message and code integrity and can be used for supporting both high speed secure networking and protection against malware in next generation computing systems.

The document is structured as follows: In Section 2 we describe the design methodology of the Vortex family. In Section 3 we describe the algorithm. In Section 4 we describe our experiments and present qualitative arguments why we believe the Vortex family is secure. Finally in Section 5 we provide some concluding remarks.

2 Design Methodology of the Vortex Family

Vortex represents a new family of one way hash functions that can produce message digests of 256 bits. The main idea behind the design of these hash functions is to use known algorithms that can support very fast diffusion in a small number of steps. Our intent is to allow each bit of an input block to affect all bits of a hash after a small number of computations.

The algorithms we use in our design are:

- The AES round due to its capability to perform very fast mixing across 32-bits as a stand-alone operation and 128-bits if combined with at least one more round; and:
- A variant of Galois Field (GF(2)) multiplication due to its capability to cross mix bits of different sets (i.e., the input operands) in a manner that is cryptographically stronger than other simpler schemes (e.g., Feistel reordering proposed in modes like MDC-2 [3]).

We also balance the cryptographic strength that comes from iterating block cipher rounds with SBox substitution and diffusion (like Whirlpool) against the need to have a lightweight implementation with as small number of rounds as possible. We use only 3 AES rounds with stronger key schedule. The design choice of 3 comes from

the fact that 2 rounds is the bare minimum number needed for 128-bit wide mixing. The authors are aware that 3 round AES transformations can be distinguished from random permutations in several ways. So there may exist properties that can be exploited for collision attacks. Our design, however, introduces a new key schedule algorithm that potentially compensates for the security lost from reducing the number of AES rounds. In any case our design is open for introducing more rounds if this is proven necessary in the future. AES rounds are followed by our variant of Galois Field multiplication. This achieves cross-mixing between 128-bit sets. Our transformation is not simple carry-less multiplication but combines bit reordering operations, XORs and additions with carries. In this way our variant of Galois Field multiplication:

- achieves better diffusion than the straightforward carry-less multiplication between the 128-bit inputs
- is a non-commutative operation protecting against attacks based on swapping the order of the chaining variables in the processing of a message.

Our family of one way hash functions uses the AES round as specified in the standard FIPS-197.

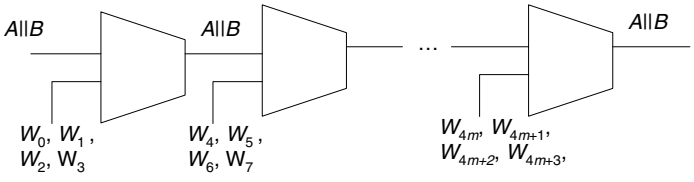


Fig. 1. Vortex as a Merkle-Damgård construction

3 Algorithm Description

Vortex processes an input stream as a sequence of 512-bit blocks. The stream is padded with a '1'. If the length of the stream is not a multiple of 512 minus 96, then the stream is further padded with zeros following the bit equal to '1'. The last 96 bits indicate the configuration of the hash (32 bits) and the length of the stream (64 bits). Each block is divided into two sub-blocks of 256 bits each and each sub-block is divided into two words of 128 bits each. Vortex operates on two 128-bit variables A and B initialized to some constant values. It processes each block using AES rounds that modify the values of A and B. In the end it returns the concatenation of A and B $A\|B$ .The algorithm for processing a sub-block is the following:

Vortex sub-block(A, B, W_0, W_1)
{
; W_0, W_1 be the words of the current sub-block to be processed
$A \leftarrow \tilde{A}_{W_0}(A)$
$B \leftarrow \tilde{A}_{W_1}(B)$
$A\|B \leftarrow V_M^{(A)}(A,B)$
return(A, B)
}

The algorithm for processing a block is the following:

Vortex block(A, B, W_0, W_1, W_2, W_3)
{
$(A, B) \leftarrow (A, B) \oplus$ Vortex sub-block(A, B, W_0, W_1) ; uses W_0 and W_1
$(A, B) \leftarrow (A, B) \oplus$ Vortex sub-block(A, B, W_2, W_3) ; uses W_2 and W_3
}

As one can see the Vortex block is essentially a Merkle-Damgård construction. It accepts a chaining variable $A\|B$ and four input words W_0, W_1, W_2, W_3 and returns an updated value of the chaining variable $A\|B$. Such construction in shown in Figure 1.

The other aspect that can be observed is that Vortex block incorporates a Davies-Meyer structure around the Vortex sub-block in order to make the transformation non-reversible. Such structure is repeated twice as shown in Figure 2:

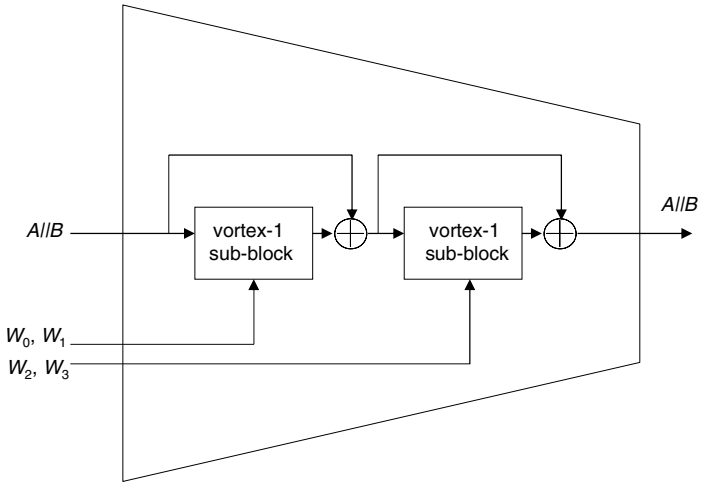


Fig. 2. Davies-Meyer structure of the Vortex block

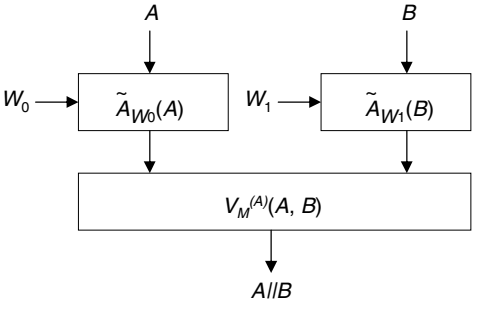


Fig. 3. Vortex sub-block

The Vortex sub-block is built upon two mathematical functions: The transformation $\tilde{A}_K(x)$ which is a lightweight block cipher and the merging function $V_M^{(A)}(A,B)$. There are two instances of the transformation $\tilde{A}_K(x)$ in the Vortex sub-block. Each instance processes a different chaining variable among A, B. Each instance of the transformation $\tilde{A}_K(x)$ treats its input chaining variable as a plaintext and its input word, which is one from W_0, W_1, W_2, W_3, as a key as it is the norm in the Davies-Meyer structure. The merging function $V_M^{(A)}(A,B)$ combines the outputs of the two instances of $\tilde{A}_K(x)$ into the new value of $A \| B$. The structure of the Vortex sub-block is shown in Figure 3.

The transformation $\tilde{A}_K(x)$ is a lightweight block cipher based on an AES round that encrypts x, which is 128 bits long, using the key K. $\tilde{A}_K(x)$ uses three AES rounds as specified in the standard FIPS-197 [1]. Each AES round consists of a round key addition in GF(2), followed by an SBox substitution phase, the ShiftRows transformation and the MixColumns transformation. The key schedule algorithm used by $\tilde{A}_K(x)$ is different from that of AES. $\tilde{A}_K(x)$ uses three 128-bit wide *Rcon* values RC_1, RC_2 and RC_3 to derive three round keys RK_1, RK_2 and RK_3 as follows:

$RK_1 \leftarrow \mathsf{Perm}(\mathrm{SBox}(K \boxplus RC_1))$

$RK_2 \leftarrow \mathsf{Perm}(\mathrm{SBox}(RK_1 \boxplus RC_2))$

$RK_3 \leftarrow \mathsf{Perm}(\mathrm{SBox}(RK_2 \boxplus RC_3))$

where Perm() is a bit permutation and by '$\boxplus$' we mean addition modulo 2^{128}. The SBox transformation in the key schedule is applied on 16 bytes, i.e., 128 bits. As explained before, a single AES round performs diffusion across 32 bits. This is accomplished through the combination of the SBox and MixColumns transformations.

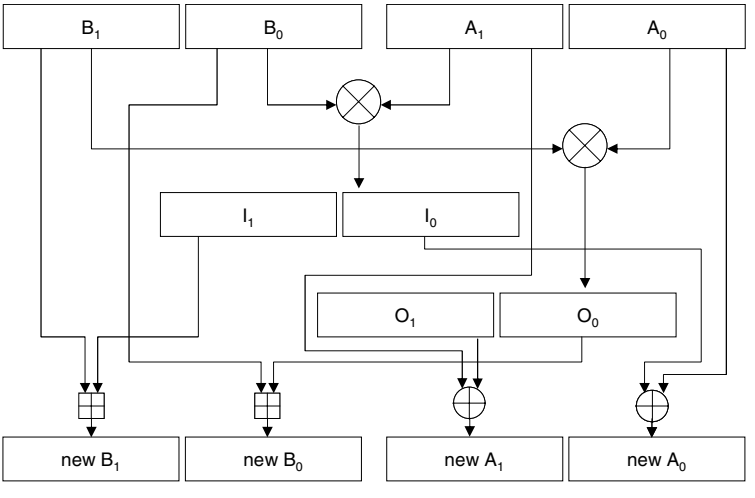


Fig. 4. The Merging Function of Vortex

Two AES rounds diffuse across 128 bits. This is accomplished through the combination of the subsequent ShiftRows and MixColumns transformations. Three rounds further strengthen the diffusion performed. The *Rcon* values can be set to some constant values (see Appendix) or derived from A and B.

The merging function, shown in Figure 4, $V_M^{(A)}(A, B)$ operates as follows:

$V_M^{(A)}(A, B)$
{
let $A = [A_1, A_0]$
let $B = [B_1, B_0]$
$O \leftarrow A_0 \otimes B_1$
$I \leftarrow A_1 \otimes B_0$
let $I = [I_1, I_0]$
let $O = [O_1, O_0]$
return $[B_1 \boxplus I_1, B_0 \boxplus O_0, A_1 \oplus O_1, A_0 \oplus I_0]$
}

where by '$\boxplus$' we mean addition modulo 2^{64}, and '$\otimes$' we mean carry-less multiplication.

The merging function is based on carry-less multiplication. Our merging function makes sure that the bits of A impact the bits of B and vice versa. In fact, each bit of one variable affects a significant number of the bits of the other variable in a non-linear manner. This makes our design better than a straightforward XOR or other simple mathematical operation. One can also observe that even though our merging function is strong cryptographically, it does not accomplish perfect mixing by itself. This is because each bit of A or B affects a large number of bits of the other variable but not all of them. Perfect mixing is accomplished by the 3 AES rounds that follow our merging function. So, for a pair of input words W_0, W_1 perfect mixing is accomplished after a sequence of 3 AES rounds (mix across 128 bits), merging using Galois Field multiplication (cross-mix across 128 bit sets but not perfect mixing) and another set of 3 AES rounds as part of the sub-block processing to follow. For this reason whereas a regular Vortex sub-block is processed using 3 AES rounds and a merging function, the last Vortex sub-block is processed using 3 AES rounds, merging, yet another 3 AES rounds and subsequent merging again.

4 Security and Performance of Vortex

The security of the Vortex family was investigated experimentally by conducting a large number of experiments (2^{20}) hashing the Vortex specification document with random perturbations, which were superimposed on it. For these experiments we computed the probability of collision and the distribution of the output differentials. Subsequently we compared the numbers we got from Vortex with numbers we got from SHA256. No collision occurred in our experiments. Our initial results indicated that there is no experimental evidence that Vortex is inferior in terms of its collision resistance and randomness of output differentials when compared to SHA256.

In fact our results can be interpreted as a strong indication that the Vortex family is at least as secure as SHA256 even though it uses smaller number of block cipher rounds. There are several reasons for this. First AES round is a good mixing function. The key used is completely data dependent and hence our scheme does not suffer from known attacks on compression functions that use a small set of keys [8]. The key schedule transformation of Vortex is stronger than AES due to the fact that the SBox transformation is applied across each 128 bit round key as opposed to 32 bits only and that round constants are added using integer addition modulo 2^{128} as opposed to XOR. It is the combination of two independent sources of non-linearities in the key schedule, i.e., addition with carries and inversion in GF(256) that potentially makes the key schedule transformation secure against differential attacks - even though the number of AES rounds is reduced for achieving better performance.

The merging function of Vortex combines linear (XORs) and non-linear (adds with carries) transformations with 64-bit carry-less multiplication building blocks. This operation is non-commutative and when combined with previous and subsequent AES rounds and Galois Field multiplication achieves perfect mixing across 256 bits. By designing the merging function to be non-commutative we destroy any symmetry in the computation of the Vortex sub-block that could be a potential source of collision. If Vortex was designed such that its merging function is commutative, then an attacker could create a collision by generating a message that swaps the position of chaining variables *A* and *B* as compared to another given message.

A more thorough analytical study on the security of the Vortex family is planned. A part of our future work we plan to develop a methodology for computing the collision resistance and the first pre-image resistance of our construction based on the divide-and-conquer approach that was first developed in the study of the MDC-2 mode by Steinberger [3]. Such approach helps with reasoning about the collision and pre-image resistance of specific components of hash functions. Components of hash functions include adders, shifters, XORs, S-Boxes, linear diffusers, bit permutations etc. Whereas our merging function is more complex than the MDC-2 mode of operation we believe that it can be potentially analyzed due to the fact that it combines relatively simple building blocks (i.e., multipliers adders and XORs). In addition multipliers are carry-less accepting small size input operands (i.e., 64 bits). These facts make the collision and pre-image resistance of our construction potentially easier to compute than MDC-2.

As another part of future work we would like to investigate the how close our lightweight block cipher is to ideal. We also need to determine the optimal relationship between the *Rcon* constants of the Vortex key schedule and the chaining variables *A*, *B*. Another question that needs answering is whether the presence of simple carry-less multiplication is sufficient in the merging function or not. Any non-zero operand multiplied with zero results in zero. Such fact can increase the collision probability associated with our merging function. If this is proven to be a design deficiency, it can be potentially corrected with simple modifications to the algorithm. For example, a single carry-less multiplication can be replaced by two multiplications. In one of the two multiplications, operands are XOR-ed with a correcting constant and the results of the multiplications are XOR-ed with each other.

We estimate that the Vortex family of algorithms can have substantial performance gain when implemented in software in future processors with support for AES round

computation and Galois Field multiplication. There are several advances in processor architecture technology as well as compact SBox implementations [12-14] that make us believe this is a trend. Expected performance is at 1.5 cycles per byte which is approximately 14X gain as compared to SHA256.

5 Concluding Remarks

We presented Vortex a new family of one way hash functions that can produce message digests of 256 bits. The main idea behind the design of these hash functions is that we use well known algorithms supporting very fast diffusion in a small number of steps. We presented a set of qualitative arguments why we believe Vortex is secure and described a set of experiments that gave us confidence that the Vortex design is not inferior to SHA256 in terms of its collision resistance and randomness of output differentials. Performance-wise the expected difference between Vortex and earlier work is substantial. SHA256 operates at 21 cycles per byte on a Core 2 Duo processor. Vortex is expected to operate at a speed of 1.5 cycles per byte in future CPUs with instruction set support for AES round computation and Galois Field (GF(2)) multiplication. We believe that the design of the Vortex family is important because it represents a scalable on-the-CPU solution for message and code integrity and can be used for supporting both high speed secure networking and protection against malware in next generation computing systems.

References

1. Advanced Encryption Standard, Federal Information Processing Standards Publication (1997) http://csrc.nist.gov/publication/fips
2. Daemen, J., Rijman, V.: The Wide Trail Design Strategy. In: Honary, B. (ed.) Cryptography and Coding 2001. LNCS, vol. 2260, pp. 222–238. Springer, Heidelberg (2001)
3. Steinberger, J.P.: The Collision Intractability of MDC-2 in the Ideal Cipher Model. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 35–41. Springer, Heidelberg (2007)
4. Knudsen, L., Lai, X., Preneel, B.: Attacks on Fast Double Block Length Hash Functions. Journal of Cryptology, No. 11, pp. 59-72, International Association for Cryptologic Research (1998)
5. Lucks, S.: Design Principles for Iterated Hash Functions, Cryptology ePrint Archive,Report 2004/253 (2004), http://eprint.iacr.org
6. Damgård, I.: A Design Principle for Hash Functions. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 416–427. Springer, Heidelberg (1990)
7. Merkle, R.: One Way Hash Functions and DES. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 428–446. Springer, Heidelberg (1990)
8. Black, J., Cochran, M., Shrimpton, T.: On the Impossibility of Highly Efficient Block Cipher-based Hash Functions. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 526–541. Springer, Heidelberg (2005)
9. Bellare, M., Ristenpart, T.: Multi-Property-Preserving Hash Domain Extension and the EMD Transform. In: Lai, X., Chen, K. (eds.) ASIACRYPT 2006. LNCS, vol. 4284, pp. 299–314. Springer, Heidelberg (2006)

10. Secure Hash Standard, Federal Information Processing Standards Publication 180-2, http://csrc.nist.gov/publication/fips
11. Menezes, A., Oorschot, P., Vanstone, S.: Handbook of Applied Cryptography. CRC Press, Boca Raton (1999)
12. Rudra, A., Dubey, P.K., Jutla, C.S., Kumar, V., Rao, J.R., Rohatgi, P.: Efficient Rijndael Encryption with Composite Field Arithmetic. In: Cryptographic Hardware and Embedded Systems - CHESS 2001, pp. 175–188 (2001)
13. Satoh, A., Morioka, S., Takano, K., Munetoh, S.: A Compact Rijndael Hardware Architecture with SBox Optimization. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248. Springer, Heidelberg (2001)
14. Moriokah, S., Satoh, A.: An Optimized S-Box Circuit Architecture for Low Power AES Design. In: Cryptographic Hardware and Embedded Systems - CHESS 2001, pp. 172–186 (2002)
15. Gueron, S., Parzanchevsky, O., Zuk, O.: Masked Inversion in GF(2n) Using Mixed Field Representations and its Efficient Implementation for AES. In: Nedjah, N., de Macedo Mourelle, L. (eds.) Embedded Cryptographic Hardwdare: Methodologies & Architectures, Nova Science Publishers, Inc (2004); (ISBN: 1-59454-012-8)

Appendix

The initial values for the chaining variables and the 3 *Rcon* constants used in our experiments are shown below. The presentation is in little endian. This means that each 32-bit word indexed by '0' is the least significant word of each 128-bit quantity. The 32-bit hash configuration value was set to zero. The bit permutation Perm() function was set to the identity function.

```
A[0]   = 0xfa32b5c2;
A[1]   = 0x235f2ad7;
A[2]   = 0x5943fa81;
A[3]   = 0x63465bcd;

B[0]   = 0x57f42acb;
B[1]   = 0x34904bde;
B[2]   = 0xe4f6a123;
B[3]   = 0xde834ba4;

RC1[0] = 0xa7523893;
RC1[1] = 0xfdea5432;
RC1[2] = 0xe45a8926;
RC1[3] = 0x37689dac;

RC2[0] = 0xf43d67b1;
RC2[1] = 0xa2d67239;
RC2[2] = 0xd90451ab;
RC2[3] = 0xe5317c26;

RC3[0] = 0xb43a7c34;
RC3[1] = 0x7c58d653;
RC3[2] = 0x6b7486f7;
RC3[3] = 0x16e875a1;
```

Comparative Evaluation of Rank Correlation Based DPA on an AES Prototype Chip*

Lejla Batina[1], Benedikt Gierlichs[1], and Kerstin Lemke-Rust[2]

[1] K.U. Leuven, ESAT/SCD-COSIC and IBBT
Kasteelpark Arenberg 10, B-3001 Leuven-Heverlee, Belgium
firstname.lastname@esat.kuleuven.be
[2] T-Systems GEI GmbH
Rabinstr. 8, 53111 Bonn, Germany
kerstin.lemke-rust@gmx.de

Abstract. We propose a new class of distinguishers for differential side-channel analysis based on nonparametric statistics. As an example we use Spearman's rank correlation coefficient. We present a comparative study of several statistical methods applied to real power measurements from an AES prototype chip to demonstrate the effectiveness of the proposed method. Our study shows that Spearman's rank coefficient outperforms all other univariate tests under consideration. In particular we note that Pearson's correlation coefficient requires about three times more samples for reliable key recovery than the method we propose. Further, multivariate methods with a profiling step which are commonly assumed to be the most powerful attacks are not significantly more efficient at key extraction than the attack we propose. Our results indicate that power models which are linear in the transition count are not optimal for the attacked prototype chip.

Keywords: Differential side-channel analysis, AES hardware, DPA, Rank correlation, Template attacks, Stochastic model.

1 Introduction

Side-channel attacks are a very active research area ever since the fundamental publications of Kocher et al. [9,10]. Especially with the evolving low-cost applications, *i.e.* pervasive security applications such as RFIDs and sensor nodes, side-channel attack resistance has become a matter of paramount importance. There are many practical attacks published and, at the same time, a firm line of work on theoretical aspects considering models for attackers, countermeasures *etc.*

It is widely believed that a correlation coefficient is the best statistical test for most power models to expose the right key among all the candidates. For

* This work was supported in part by the IAP Programme P6/26 BCRYPT of the Belgian State (Belgian Science Policy), by FWO projects G.0475.05, and G.0300.07, by the European Comission through the IST Programme under Contract IST-2002-507932 ECRYPT NoE, and by the K.U. Leuven-BOF.

T.-C. Wu et al. (Eds.): ISC 2008, LNCS 5222, pp. 341–354, 2008.

this purpose the common choice is Pearson's correlation coefficient [5] in conjunction with the Hamming weight or distance model [3]. On platforms like microcontrollers, where the relationship between the transitions on a data bus and the observable power dissipation is strikingly linear, this choice is theoretically founded. However, other parts of a microcontroller, *e.g.* registers, and different platforms such as ASICs and FPGAs do not necessarily follow this simple and linear relationship. We found that there are better matches for the function. Relaxing the assumption to simply a monotonic function led us to a new set of distinguishers based on nonparametric statistics. The results of our study show improvements with respect to efficiency, measured in the number of samples required, when we compare to the methods under consideration.

The contribution of our work is fourfold: i) We introduce a new class of side-channel distinguishers based on nonparametric statistics. We demonstrate the effectiveness of our approach by applying Spearman's rank correlation coefficient in a comparative study. ii) We show that rank correlation reaches the highest success rate amongst all univariate methods and in particular outperforms Pearson's correlation coefficient on this platform. Therefore it must be considered as an important distinguisher. iii) We give a detailed comparison of well known and adopted attacks on an AES hardware module. To the best of our knowledge the only related work was published by Mangard et al. in [13] and applied DPA to unprotected and to masked CMOS, but they varied the attacked intermediate results of AES and not the statistical distinguisher. iv) We present the first comparative study of templates and stochastic models on an AES hardware module. The work in [2] also discusses template attacks but the test platform is a DES hardware module.

This paper is organized as follows. Section 2 summarizes previous and related work. Section 3 describes the architecture of the targeted AES hardware module. In Section 4 we introduce a new class of side-channel distinguishers based on nonparametric statistics and in particular Spearman's rank correlation coefficient. Section 5 briefly explains the attacks and distinguishers used in our study. Experimental results from an unprotected prototype chip in standard CMOS (sCMOS) technology are provided in Section 6. Section 7 concludes the paper and outlines future work.

2 Previous Work

A decade ago Kocher et al. introduced successful attacks by measuring the power consumption during the execution of cryptographic algorithms [10]. It was demonstrated that one can use the physical leakage to easily recover secret keys if no countermeasures were deployed in the implementation. The demonstrated attack known as Differential Power Analysis (DPA) was applied against implementations of cryptographic algorithms running on smart cards. The surprising results gave rise to a new research area and there have been many contributions on both theoretical and practical aspects of power analysis. Other side-channels were also introduced such as as electromagnetic emanation [6,17], timing [9], acoustics [21] *etc.*

DPA attacks as introduced in [10] use a so-called selection function to sort a set of power consumption samples into subsets. The authors proposed simple boolean partitioning to divide the power samples in two subsets. However, the selection function can be extended to more bits and accordingly the power samples are sorted into multiple subsets. In this case we speak about a multi-bit DPA [14]. Selection functions are defined on an intermediate value of the cryptographic algorithm under attack that can be predicted using a key hypothesis and known data. It is afterwards a statistical question to find a key hypothesis that results in the highest correlation between the predicted values of a selection function and the sampled power consumption. Kocher et al. suggested to apply the difference of means test to find the right key. To such tests one usually refers as side-channel distinguishers. Other distinguishers referred to in the literature are Pearson's correlation coefficient [5], Mutual Information [7], Bayesian classification, e.g. template attacks introduced by Chari et al. [4] and the stochastic model by Schindler et al. [20]. Distinguishers are also sometimes used to assist other side-channel attacks. For example, Rechberger and Oswald proposed to use a DPA attack to find interesting points in time for templates in [18].

In this paper we introduce a new class of distinguishers based on nonparametric statistics, and compare them with other widely adopted techniques. We show that Spearman's rank correlation coefficient outperforms all other univariate methods under consideration on an AES ASIC implementation in sCMOS.

A similar comparative study of templates and stochastic models was performed by Gierlichs et al. [8], but they attacked an AES software implementation. To our best knowledge the only practical side-channel attacks on real AES chips were published by Örs et al. [15] and by Mangard et al. [13]. In [15] Pearson's correlation coefficient was used to perform a DPA attack on an unprotected implementation. The authors of [13] performed extended DPA by focusing on different choices for the selection function and not on statistical tests. However, DPA attacks on both unprotected and protected CMOS were performed. The important result was that the use of algorithmic masking in hardware does not increase the side channel resistance substantially in the presence of glitches. Another example where an ASIC platform was attacked can be found in [2]. The authors applied a template attack on a DES implementation focusing on the key schedule and they used a special power model.

3 Architecture of the AES Hardware Module

Our experimental platform is an AES hardware module from the SCARD chip. The chip is an outcome of the "Side-Channel Analysis Resistant Design Flow - SCARD" project led by the European Commission [22]. It contains an 8051 microcontroller with AES-128 co-processor in 0.13 μm sCMOS and several secured logic styles.

In the sequel we focus on the AES module which is implemented in standard CMOS logic and includes no countermeasures against side-channel attacks. The AES module supports AES-128 [1] encryption and decryption in ECB mode.

The implementation uses four parallel one-stage pipelined implementations of the AES S-Box. A similar implementation is described in [12]. The module includes the following parts: data unit, key unit, and interface. The most important part is the data unit (see Fig. 1), which includes the AES operation. It is composed of 16 data cells ($C_{i,j}$, where $i, j \in \{0, 1, 2, 3\}$) and four S-Boxes. A data cell consists of flip-flops (able to store 1 byte of data) and some combinational logic in order to perform AddRoundKey operations. Load data is done by shifting the input data column by column into the registers of the data cells. The initial AddRoundKey transformation is performed in the fourth clock cycle together with the load of the last column. To calculate one round, the bytes are rotated vertically to perform the S-box and the ShiftRows transformation row by row. In the first clock cycle, the S-Box transformation starts only for the fourth row. Because of pipelining the result is stored two clock cycles later in the first row. S-boxes and the ShiftRows transformations can be applied to all 16 bytes of the state within five clock cycles due to pipelining. The architecture is very compact and suitable for smartcards and other wireless applications, which makes the attacks extremely relevant.

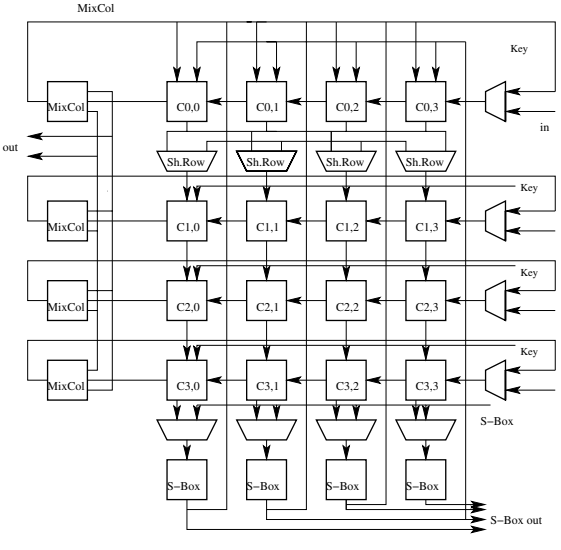


Fig. 1. The architecture of the AES module

The S-Boxes in the AES module are implemented by using composite field arithmetic, *i.e.* $GF(2^8)$ is considered as an extension field of $GF(2^4)$ as proposed by Wolkerstorfer et al. [23]. The original idea comes from Rijmen [19] as he suggested using subfield arithmetic in the crucial step of computing an inverse in the Galois Field.

We note here that the specifics of architecture do not cause the effectiveness of the attack proposed. The only fact about the platform that our distinguisher

takes advantage of is that the power model is not strictly linear in the transition count. This results in the attack performing better than other known methods.

4 Rank Correlation

Here we discuss some techniques which are usually referred to as nonparametric statistics [11] in the literature. Nonparametric equivalents to the standard correlation coefficient (*i.e.* Pearson's ρ) are Spearman's ρ, Kendal's τ, and Γ coefficient. These are also sometimes called nonparametric correlation coefficients. We demonstrate that in our experiments Spearman's correlation coefficient performs much better than the one of Pearson. This result suggests that one should consider alternative statistical tests in order to improve an attack's efficiency with respect to the number of required samples. This issue is also heavily platform-related so the influence of a power model is the most relevant one.

Figure 2 (left) shows the mean and the standard deviation of the power consumption as a function of the Hamming weight derived from a microcontroller moving data over its internal bus. The graph indicates that the relationship between power and the data's Hamming weight is very close to linear and that the empirical standard deviation is low. The plot on the right side of Fig. 2 on the other hand shows that the dependency between power consumed by a register update in the AES module and the Hamming distance of two subsequently stored data words, *i.e.* transition count, is not so close to linear. It is linear over small intervals but overall we can only say that it is a monotonic function. The large standard deviation can be caused either by algorithmic noise, *i.e.* it could reflect the power dissipation of the processing of other bits in parallel, or it can indicate that register updates with identical transition count do not lead to similar power consumption. The graph suggests that there might be a more suitable model than the strictly linear one. In general, one speaks also about the level of measurement, which can be interval, ordinal etc. In that case, one should look into nonparametric statistics, *i.e.* rank correlation, instead of Pearson's correlation coefficient.

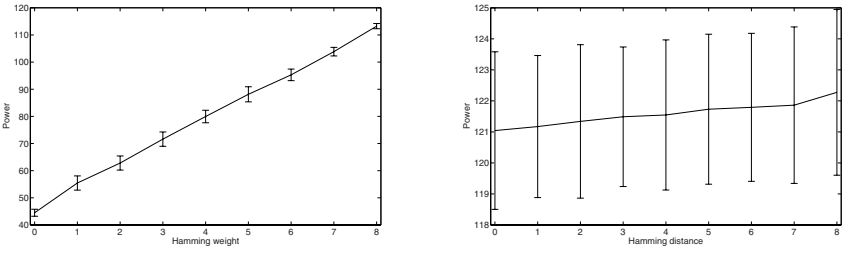


Fig. 2. Power dissipation of a micro-controller when accessing the data bus over Hamming weight (left); Power dissipation of the AES module when updating an 8bit register over Hamming distance (right)

The Pearson correlation coefficient is also known as the product-moment coefficient and it shows linear fits to (sometimes) noisy data. Pearson's ρ requires more information in the data than Spearman's coefficient, because it assumes the data is interval or ratio scaled, while Spearman's coefficient only expects it to be ordinal scaled. Data measured at the interval level are called interval scaled data, and data given with rank orders are called ordinal variables or rank variables.

Unlike Pearson's ρ, rank correlation does not assume a linear relationship between variables. A nonparametric (distribution-free) rank statistic is proposed by Spearman in 1904 and it can be also used as test of independence between two variables. More precisely, it is a measure of the strength of the association between two variables [11]. The Spearman rank correlation coefficient is a measure of monotonic relationship, which means that it can be used also if the relationship is non-linear. It was mainly meant to be used when the distribution of the data make Pearson's correlation coefficient unsuitable or misleading.

Let n be the number of pairs of values for variables X and Y defined on the (discrete) spaces $\mathcal{X}$ and $\mathcal{Y}$ and let d_i be the difference between each rank of corresponding values of X and Y. The formula to compute Spearman's rank correlation is:

$$\rho = 1 - \frac{6\sum_i {d_i}^2}{n(n^2-1)} \quad \text{or} \quad \rho = \frac{\sum_i (R_i - \overline{R})(S_i - \overline{S})}{\sqrt{\sum_i (R_i - \overline{R})^2 (S_i - \overline{S})^2}}, \tag{1}$$

where R_i and S_i are the ranks of variables X and Y. The latter formula is preferable if tied ranks exist, *i.e.* if the data to be ranked contains more than one value. In this case, Spearman's coefficient is actually computed as Pearson's correlation between ranks. The limited computational overhead is therefore given by the ranking process.

In Sect. 6 we show that this new side-channel distinguisher performs much better than Pearson's coefficient on our CMOS AES module. This is likely due to the specifics in the power consumption properties of the device.

Spearman's coefficient is, however, still insensitive to some types of dependence. Kendall's rank correlation gives a better measure of correlation and is also a better two sided test for independence. The Gamma statistic is preferable to both, Spearman or Kendall when the data contains many tied observations, but comes with the cost of increased computational complexity.

5 Established Side-Channel Attacks and Distinguishers

In this section we briefly recall known attacks which we apply to the AES hardware module.

5.1 Single-bit and Multi-bit DPA

(Single-Bit) DPA as proposed in [10] computes the DPA bias signal

$$\Delta_t = \frac{\sum_i p_{i,t} l_i}{\sum_i l_i} - \frac{\sum_i p_{i,t}(1-l_i)}{\sum_i (1-l_i)} \tag{2}$$

as the difference between the average of all measurements for which the so called selection function l_i evaluates to 1 and the average of all measurements for which the selection function evaluates to 0. The summations are taken over the q samples and the bias signal has to be computed for each time slice t within the power measurements p.

In [14] Messerges proposes to use selection functions based on several bits of the targeted intermediate value. He suggests to compute the DPA bias signal from the two subsets of power samples for which the selection function evaluates to maximal distance. For a selection function considering three bits for example, one would compute the difference of means of the subset "000" and the subset "111".

5.2 Pearson Correlation

In [3] Brier et al. suggest to estimate the Pearson correlation coefficient between a vector of power consumption samples p and a vector of power consumption predictions l

$$\rho_t = \frac{q\sum_i p_{i,t}l_i - \sum_i p_{i,t}\sum_i l_i}{\sqrt{q\sum_i p_{i,t}^2 - (\sum_i p_{i,t})^2}\sqrt{q\sum_i l_i^2 - (\sum_i l_i)^2}}\,. \tag{3}$$

The summations are taken over the q measurements and the correlation coefficient has to be estimated for each time slice t within the power curves p.

5.3 Multivariate Analysis

For multivariate analysis, it is assumed that the measurement vector $\boldsymbol{z} \in \mathbb{R}^m$ is distributed according to an m-variate Gaussian density

$$\mathcal{N}(\boldsymbol{z}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{(2\pi)^m|\boldsymbol{\Sigma}|}} \exp\left[-\frac{1}{2}(\boldsymbol{z}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{z}-\boldsymbol{\mu})\right] \tag{4}$$

where $\boldsymbol{\mu}$ is the mean vector, $\boldsymbol{\Sigma}$ the covariance matrix of the normally distributed random variable $\boldsymbol{Z}$ with $\boldsymbol{\Sigma} = (\sigma_{uv})_{1\le u,v\le m}$ and $\sigma_{uv} := \mathbb{E}(Z_uZ_v) - \mathbb{E}(Z_u)\mathbb{E}(Z_v)$, $|\boldsymbol{\Sigma}|$ denotes the determinant of $\boldsymbol{\Sigma}$ and $\boldsymbol{\Sigma}^{-1}$ its inverse. A Gaussian distribution is completely determined by its parameters $(\boldsymbol{\mu}, \boldsymbol{\Sigma})$. Both parameters can depend on the data processed, therefore enabling side channel leakage.

Both template attacks as well as stochastic methods consist of two-stages with different assumptions. The first stage is a profiling phase at which both key and plaintext or ciphertext are assumed to be known to the adversary. As result of profiling, the adversary obtains an m-variate Gaussian characterization of the key dependent physical leakage. The second stage is the key recovery stage (or classification) at which the adversary knows the plaintext or ciphertext, but not the key. At the second stage, the adversary's objective is key recovery.

Template Attacks. Roughly summarizing, there are three steps for building templates in the profiling stage. Firstly, the adversary computes the mean vector

$\boldsymbol{\mu}_k$ for each key dependency k. Secondly, m points in time are selected where significant differences are recognized among the mean vectors for different key dependencies. Finally, for each key dependency k the m-variate estimation of the noise is carried out resulting in the Gaussian distribution $\mathcal{N}(\boldsymbol{z}, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$. For a more detailed description of the algorithms we refer to [4,18,8].

Template classification computes the maximum likelihood, i.e., given n' measurements the adversary decides for the key hypothesis k^* that maximizes

$$\alpha_k := \prod_{i=1}^{n'} \mathcal{N}(\boldsymbol{z}_i, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \tag{5}$$

among all k. Note that for practical purposes the log-likelihood is more adequate.

Stochastic Methods. Stochastic methods are an alternative approach for m-variate side channel analysis and have been introduced in [20] from which we only consider the so called 'maximum likelihood principle' in this paper.

In contrast to templates, stochastic methods estimate only one covariance matrix $\boldsymbol{\Sigma}$ that is used for all key dependent Gaussian densities $\mathcal{N}(\boldsymbol{z}, \boldsymbol{\mu}_k, \boldsymbol{\Sigma})$. Furthermore, stochastic methods estimate the mean vector $\boldsymbol{\mu}_k$ by using general linear least squares targeting one key dependent and predictable intermediate result of the cryptographic implementation based on a power model. The power model used determines the vector subspace for the linear regression. Besides the Hamming weight model, a common power model is the bit-wise coefficient model saying that each bit of an intermediate result contributes to the overall power consumption. For example, for an 8-bit data item one uses a nine-dimensional vector subspace, spanned by the constant function 1 and eight single bits of the data item in the bit-coefficient model and a two-dimensional vector subspace spanned by the constant function 1 and the Hamming weight of the data item in the Hamming weight model. For a more detailed explanation of the applied algorithms at profiling we refer to [20,8].

Classification computes the maximum likelihood, i.e., given n' measurements the adversary decides for the key hypothesis k^* that maximizes

$$\alpha_k := \prod_{i=1}^{n'} \mathcal{N}(\boldsymbol{z}_i, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}) \tag{6}$$

among all k. As the covariance matrix $\boldsymbol{\Sigma}$ is identical, this is equivalent to minimizing the term $\sum_{i=1}^{n'} (\boldsymbol{z}_i - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{z}_i - \boldsymbol{\mu}_k)$.

6 Experimental Results

Our experimental platform is the sCMOS AES hardware module from the SCARD chip. The architecture of the AES co-processor is discussed in detail in Sect. 3. We obtained 50 000 power measurements p_i $(i = 1, \ldots, 50\,000)$ by sampling the voltage drop over a 50Ω resistor inserted in the chip's Vdd line at

a rate of 2 GS/s while the coprocessor was encrypting randomly chosen plaintext messages.

Let $x_i \in \{0,1\}^8$ ($i \in \{0,1,\ldots,15\}$) denote the plaintext byte. Accordingly, let $k_i \in \{0,1\}^8$ be the corresponding AES key byte. By $S(\cdot)$ we denote the AES S-box. The intermediate result chosen is

$$\Delta_{ii'} = S(x_i \oplus k_i) \oplus S(x_{i'} \oplus k_{i'}) \tag{7}$$

with $i \neq i'$. This intermediate result $\Delta_{ii'}$ is for example given by the differential of two adjacent data cells in the studied AES hardware architecture. $\Delta_{ii'}$ depends on two 8-bit inputs to the AES S-box $(x_i \oplus k_i, x_{i'} \oplus k_{i'})$. For the comparison of statistical tests, the targeted data cells are C0,0 and C0,1 of Fig. 1 in the remainder.

6.1 Difference of Means

The difference of means distinguisher failed at our scenario. We tested single-bit and multi-bit (two, three, and four bits) selection functions and considered up to $q = 25\,000$ power samples. No parameter combination led to key discovery.

6.2 Correlation Coefficients

Figure 3 shows the results we obtain for Pearson's and Spearman's correlation coefficient when using $q = 50\,000$ measurements and the correct key hypothesis. An attack with all 2^{16} key hypotheses still indicates the two correct key bytes when we reduce the number of measurements to $q = 5000$. Therefore we use at most 5000 measurements for the following comparison of Pearson's and Spearman's coefficient. To reduce computational complexity we assume in the remainder that the key byte k_i is known and test, whether the correct value of key byte $k_{i'}$ can be recovered. The number of key hypotheses is reduced to 2^8.

Figure 4 shows the efficiency of Pearson's correlation coefficient in detecting the correct key value from a given number q of power samples. We plot the maximum positive and minimum negative correlation (y-axis) over the number q of samples (x-axis) that we obtained for each key hypothesis on the overall time section. The correlation trace for the correct key hypothesis is plotted in

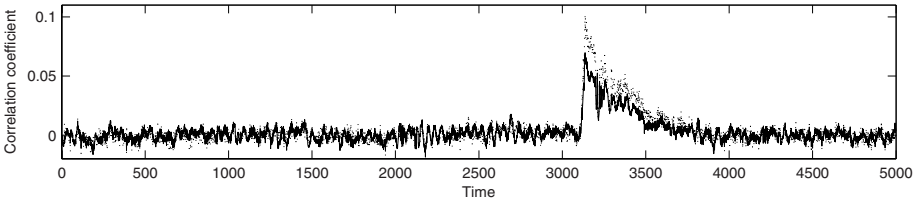


Fig. 3. Correlation coefficients for 8-bit Hamming distance as a function of time. Pearson (solid) and Spearman (dotted).

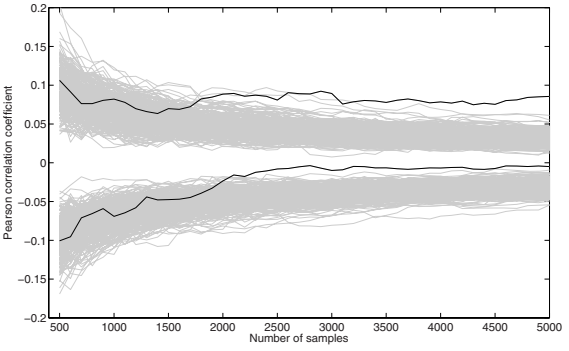


Fig. 4. Min and max Pearson correlation coefficient over number of samples; the black traces correspond to the correct key hypothesis

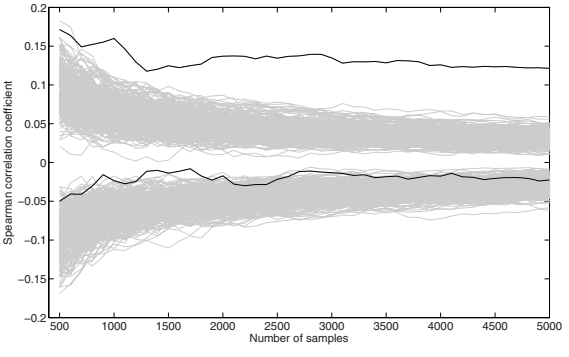


Fig. 5. Min and max Spearman Rank correlation coefficient over number of samples; the black traces correspond to the correct key hypothesis

black. One can observe that approximately $q = 4000$ power samples are required for key recovery.

Figure 5 depicts the performance of the Spearman rank correlation coefficient in the same manner as for Fig. 4. Obviously, significantly less samples (about $q = 1300$ or roughly 30% of the measurements needed by Pearson's correlation coefficient) are required for key recovery. Note that all numbers in this comparison have been confirmed by an experiment with a second data set and targeting a different cell in the hardware architecture. We report on the attacks' success rates as a function of the number of measurements in Sect. 6.5.

6.3 Stochastic Methods

Stochastic methods are applied in the bit-wise coefficient model, *i.e.* a nine-dimensional vector subspace is used for the estimation of the intermediate result

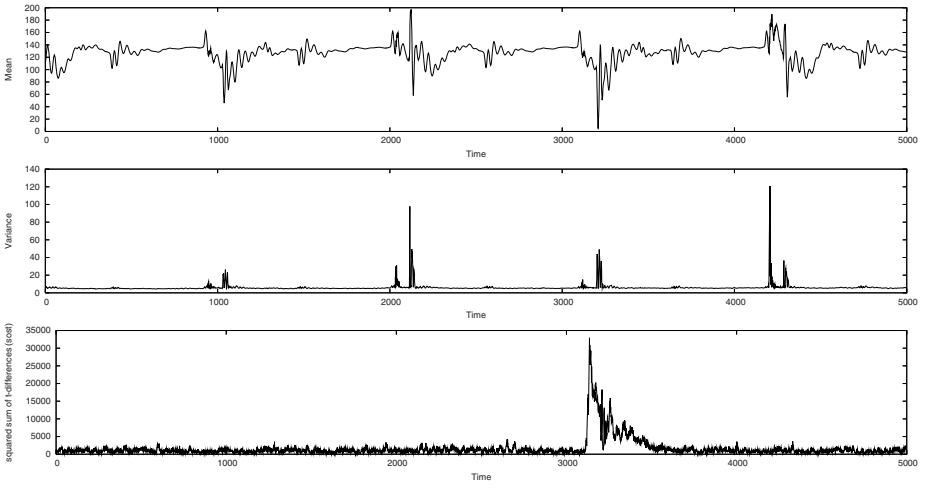


Fig. 6. Average curve (top), variance curve (middle) and sost curve (bottom) for the relevant time section derived from 40 000 measurements

in (7). For the profiling phase and classification phase we use complementary sets of measurements. In total, 40 000 measurements are used for profiling and 10 000 measurements for classification purposes.

Fig. 6 shows the mean and variance vector in the time frame for which we observed correlation peaks in the previous experiments. We chose the squared sum of t-differences (*sost*) trace (cf. [8]) for the identification of contributing points in time that is also shown in Fig. 6. For the computation of the *sost* trace the data dependent coefficients for the intermediate result (7) were estimated with 40 000 measurements. As result of this estimation one can compute the mean vector for each possible value of (7).

After identification of points of interest, the estimation of the mean vectors is repeated with 20 000 measurements and the estimation of the covariance matrix at the selected points in time is done with the other disjunctive set of 20 000 measurements.

Classification success rates are about 73% for $n' = 1000$ measurements, 97% for $n' = 2000$ measurements, and 100% for $n' = 3000$ measurements using ten selected points of interest ($m = 10$).

6.4 Template Attack

As for the stochastic method, we use a set of $n = 40\,000$ measurements for the profiling phase and a complementary set of $n' = 10\,000$ measurements for the classification phase. After the estimation of the mean vectors $\boldsymbol{\mu}_k$ we compute the *sost* trace (cf. [8]) which indicates interesting points in time. As one can see in Fig. 7 (left) the *sost* trace points toward a very narrow time window. The *sost* trace within this time window, see Fig. 7 (right), looks very similar to

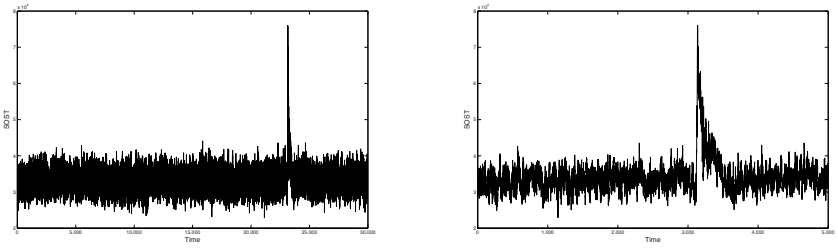


Fig. 7. *sost* trace template attack for the overall (left) and the relevant time frame (index 20 000 to 25 000, right)

the *sost* trace shown in Fig. 6. Again we experiment with the number and the distribution of points of interest. For the sake of comparison we report about the best results we could achieve using $m = 10$ points.

Once the points of interest are chosen, we estimate the covariance matrices $\mathbf{\Sigma}_k$ from the same set of 40 000 measurements. It turns out that classification of samples from the remaining set of measurements leads only to negligible success rates. We assume that the failure is caused by the number of measurements we use. If the number of measurements is too small, the estimations of the $\boldsymbol{\mu}_k$ and in particular of the $\mathbf{\Sigma}_k$ are bad. Since the stochastic method achieves reasonable success rates, we decide to estimate only a single, key-independent covariance matrix $\mathbf{\Sigma}$. But again, the template attack achieves only minor success rates. For a final test, we follow the suggestion of [16] and do not estimate the covariance matrix $\mathbf{\Sigma}$ at all, but simply set it to the unity matrix. This choice reflects the assumption that the side-channel leakage at the selected points in time is independent. This setting leads to classification success rates of about 32% for $n' = 1000$ samples, 63% for $n' = 2000$ samples, and 82% for $n' = 3000$ samples.

6.5 Overall Comparison

The complete results for the comparison are given in Table 1. The success rates refer to various numbers of measurements, ranging from 500 to 3000 curves, that were used for an attack and are derived from 500 experiments each using a set of randomly chosen measurements. It is obvious that Spearman's coefficient outperforms all other univariate distinguishers in all cases.

When comparing the performance of the template attack and the stochastic method, we conclude that in our scenario the stochastic method leads to better success rates and is the method of choice. The authors of [8] observed that the stochastic method can lead to better results than the template attack if the number of measurements for the profiling step is not sufficiently large. To enable the template attack on this AES hardware module, key-dependent covariance matrices $\mathbf{\Sigma}_k$ need to be replaced with a single matrix $\mathbf{\Sigma}$ and furthermore this matrix has to be set to the identity map. This fact might deserve further research

Table 1. Success rates for the distinguishers for the given number of measurements: distance of means, Pearson correlation, Spearman rank correlation, template attack with a single covariance matrix set to the unity matrix, stochastic model

No.	DoM	Pearson Corr.	Sp. Rank. Corr.	Template Attack	Stochastic Model
500	-	13.6%	39.6%	15.6%	41.4%
1000	-	29.8%	77.8%	31.8%	73.4%
2000	-	64.2%	99.0%	63.2%	96.8%
3000	-	84.0%	100.0%	82.4%	100%

on the application of template attacks if the target of evaluation is a hardware module. A more detailed investigation of this matter is beyond the scope of this paper but will be part of our future work.

7 Conclusions

We propose a new class of side-channel distinguishers based on nonparametric statistics. We compare the efficiency of Spearman's rank correlation coefficient to that of other known attack methods when extracting the key from an AES-128 prototype chip. The results allow two conclusions. Spearman's rank correlation coefficient performs best amongst the univariate methods we apply. In particular, it outperforms Pearson's correlation coefficient by far, requiring only about 30% of the number of samples. This observation indicates that a power model which is linear in the transition count is suboptimal. The observation is naturally bound to the targeted device and different platforms can lead to different results. Moreover, multivariate methods with a profiling step which are commonly considered the most powerful attacks require much more measurements and do not perform significantly better than the proposed distinguisher in this experiment. A detailed investigation of this matter is beyond the scope of this paper, but part of our future research.

References

1. FIPS 197: Announcing the Advanced Encryption Standard (AES) (November 2001), http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf
2. El Aabid, M.A., Guilley, S., Hoogvorst, P.: Template Attacks with a Power Model. Cryptology ePrint Archive, Report, 2007/443 (2007), http://eprint.iacr.org/
3. Brier, E., Clavier, C., Olivier, F.: Correlation power analysis with a leakage model. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 16–29. Springer, Heidelberg (2004)
4. Chari, S., Rao, J.R., Rohatgi, P.: Template Attacks. In: Kaliski, B.S., Koç, Ç., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 13–28. Springer, Heidelberg (2003)
5. Coron, J.-S., Kocher, P.C., Naccache, D.: Statistics and secret leakage. In: Frankel, Y. (ed.) FC 2000. LNCS, vol. 1962, pp. 157–173. Springer, Heidelberg (2001)
6. Gandolfi, K., Mourtel, C., Olivier, F.: Electromagnetic Analysis: Concrete Results. In: Koç, Ç.K., Naccache, D., Paar, C. (eds.) CHES 2001. LNCS, vol. 2162, pp. 251–261. Springer, Heidelberg (2001)

7. Gierlichs, B., Batina, L., Tuyls, P., Preneel, B.: Mutual Information Analysis - A Generic Side-Channel Distinguisher. In: Oswald, E., Rohatgi, P. (eds.) CHES 2008. LNCS, vol. 5154, pp. 426–442. Springer, Heidelberg (2008)
8. Gierlichs, B., Lemke-Rust, K., Paar, C.: Templates vs. Stochastic Methods. In: Goubin, L., Matsui, M. (eds.) CHES 2006. LNCS, vol. 4249, pp. 15–29. Springer, Heidelberg (2006)
9. Kocher, P.C.: Timing attacks on implementations of Diffie-Hellman, RSA, DSS and other systems. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 104–113. Springer, Heidelberg (1996)
10. Kocher, P.C., Jaffe, J., Jun, B.: Differential Power Analysis. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999)
11. Lehman, E.L., D'Abrera, H.J.M.: Nonparametrics: Statistical Methods Based on Ranks. Prentice-Hall, Englewood Cliffs (1998)
12. Mangard, S., Aigner, M., Dominikus, S.: A Highly Regular and Scalable AES Hardware Architecture. IEEE Trans. Computers 52(4), 483–491 (2003)
13. Mangard, S., Pramstaller, N., Oswald, E.: Successfully Attacking Masked AES Hardware Implementations. In: Rao, J.R., Sunar, B. (eds.) CHES 2005. LNCS, vol. 3659, pp. 157–171. Springer, Heidelberg (2005)
14. Messerges, T.S.: Power Analysis Attacks and Countermeasures on Cryptographic Algorithms. PhD thesis (2000)
15. Örs, S.B., Gürkaynak, F., Oswald, E., Preneel, B.: Power-analysis attack on an ASIC AES implementation. In: Proceedings of the International Conference on Information Technology (ITCC), Las Vegas, NV, USA, April 5-7 (2004)
16. Oswald, E., Mangard, S.: Template Attacks on Masking – Resistance is Futile. In: Abe, M. (ed.) CT-RSA 2007. LNCS, vol. 4377, pp. 243–256. Springer, Heidelberg (2006)
17. Quisquater, J.-J., Samyde, D.: ElectroMagnetic Analysis (EMA): Measures and Couter-Measures for Smard Cards. In: Attali, I., Jensen, T.P. (eds.) E-smart 2001. LNCS, vol. 2140, pp. 200–210. Springer, Heidelberg (2001)
18. Rechberger, C., Oswald, E.: Practical Template Attacks. In: Lim, C.H., Yung, M. (eds.) WISA 2004. LNCS, vol. 3325, pp. 440–456. Springer, Heidelberg (2005)
19. V. Rijmen.: Efficient Implementation of the Rijndael SBox, http://www.iaik.tugraz.at/RESEARCH/krypto/AES/old_rijmen/rijndael/sbox.pdf
20. Schindler, W., Lemke, K., Paar, C.: A Stochastic Model for Differential Side Channel Cryptanalysis. In: Rao, J.R., Sunar, B. (eds.) CHES 2005. LNCS, vol. 3659, pp. 30–46. Springer, Heidelberg (2005)
21. A. Shamir, E. Tromer.: Acoustic cryptanalysis, http://theory.csail.mit.edu/~tromer/acoustic/
22. The SCARD project, http://www.scard-project.eu/
23. Wolkerstorfer, J., Oswald, E., Lamberger, M.: An ASIC Implementation of the AES SBoxes. In: Preneel, B. (ed.) CT-RSA 2002. LNCS, vol. 2271, pp. 67–78. Springer, Heidelberg (2002)

Collisions for RC4-Hash*

Sebastiaan Indesteege[1,2,**] and Bart Preneel[1,2]

[1] Department of Electrical Engineering ESAT/SCD-COSIC,
Katholieke Universiteit Leuven. Kasteelpark Arenberg 10, B-3001 Heverlee, Belgium
{sebastiaan.indesteege,bart.preneel}@esat.kuleuven.be
[2] Interdisciplinary Institute for BroadBand Technology (IBBT), Belgium

Abstract. RC4-Hash is a variable digest length cryptographic hash function based on the design of the RC4 stream cipher. In this paper, we show that RC4-Hash is not collision resistant. Collisions for any digest length can be found with an expected effort of less than 2^9 compression function evaluations. This is extended to multicollisions for RC4-Hash. Finding a set of 2^k colliding messages has an expected cost of $2^7 + k \cdot 2^8$ compression function evaluations.

Keywords: RC4-Hash, hash functions, collisions, multicollisions.

1 Introduction

Cryptographic hash functions have been receiving much attention from the cryptologic community recently, as several of the widely used hash functions like MD5, SHA-0 and SHA-1, have been broken, or at least shown to be weaker than expected [3,9,10,11]. This is a motivation for the design of new hash functions, based on different design principles. One such proposal is RC4-Hash, which was introduced by Chang, Gupta and Nandi [1] in 2006. The design is inspired by the RC4 stream cipher. The latter was designed by Ron Rivest in 1987, but remained a trade secret until it leaked out in 1994 [8]. The motivation for basing a hash function design on RC4, which is well studied, is to be able to use existing results on RC4 in the security analysis of RC4-Hash [1]. Concerning the performance of RC4-Hash, the designers claim that SHA-1 is roughly 1.5 times faster than RC4-Hash [1].

We focus on the collision resistance of RC4-Hash. Informally, collision resistance means that it should be hard to find two distinct messages $m \neq m'$ that hash to the same value, *i.e.*, $h(m) = h(m')$. We show that RC4-Hash is not collision resistant, and give a method to find colliding message pairs with an expected time complexity of less than 2^9 compression function evaluations. We also extend this to multicollisions.

* This work was supported in part by the IAP Programme P6/26 BCRYPT of the Belgian State (Belgian Science Policy), and in part by the European Commission through the IST Programme under Contract IST-2002-507932 ECRYPT.

** F.W.O. Research Assistant, Fund for Scientific Research — Flanders (Belgium).

T.-C. Wu et al. (Eds.): ISC 2008, LNCS 5222, pp. 355–366, 2008.

This paper is organised as follows. In Sect. 2, a short description of the RC4-Hash family of cryptographic hash functions is given. Section 3 introduces two distinct methods to construct fixed points of the internal state of RC4-Hash. This is then used in Sect. 4 to construct colliding message pairs for RC4-Hash. In Sect. 5, extensions of the attack, as well as ways to mitigate it, are discussed. Section 6 concludes.

2 Description of RC4-Hash

RC4-Hash follows the "wide pipe" hash function design principle proposed by Lucks [7], which implies that the intermediate state size is (much) larger than the digest size. More specifically, RC4-Hash consists of a compression function $\mathcal{C} : \{0,1\}^w \times \{0,1\}^m \mapsto \{0,1\}^w$, and an output transformation $g_n : \{0,1\}^w \mapsto \{0,1\}^n$. The intermediate state size w is (much) larger than the digest length n. The compression function $\mathcal{C}$ is applied iteratively for every (padded) message block of length m, starting from an initial value. Then, the output transformation g compresses the large internal state down to the required digest length n.

In RC4-Hash, the intermediate state consists of an array S of 256 bytes and a pointer into this array, denoted by j. The array S always represents a permutation of the numbers 0 to 255. The size of the internal state is thus $\log_2(2^8!) + 8 \approx 1692$ bits. The digest length is variable from 16 bytes to 64 bytes, which is much shorter than the internal state size. The length of the message blocks is fixed to 64 bytes.

Padding Rule. A message M is padded in the following way. The 8-bit binary representation of the digest length n (in bytes), $\mathrm{bin}_8(n)$, is prepended to the message. A single "1" bit, v "0" bits and the 64-bit binary representation of the original message length (in bits), $\mathrm{bin}_{64}(|M|)$, are appended to the message. The number v is the least non-negative integer such that $|M| + 73 + v \equiv 0 \bmod 512$. This ensures that the padded message length is an integer multiple of 512 bits, the message block length. Hence, the padded message can be split into t blocks of 512 bits each, denoted by M_1 through M_t.

$$\mathrm{pad}(M) = \mathrm{bin}_8(n) \,||\, M \,||\, 1 \,||\, 0^v \,||\, \mathrm{bin}_{64}(|M|) = M_1 || M_2 || \cdots || M_t \quad . \tag{1}$$

Compression Function. The compression function of RC4-Hash, which is denoted by $\mathcal{C}\big(\langle S, j\rangle, X\big)$, is described in Fig. 1. It updates the internal state $\langle S, j\rangle$ in 256 steps. In every step, the pointer j is updated using one byte of the message block X. Then, two elements of the array S are swapped. Each of the 64 bytes of the message block is used in four steps. The order in which they are used is given by the message reordering $r(\cdot)$, see Table 5. This compression function is applied iteratively for every message block M_1 through M_t, starting from the initial state $\langle S^{\mathrm{IV}}, 0\rangle$. The initial value permutation S^{IV} is given in Table 6.

Output Transformation. After every block of the padded message has been processed, an output transformation $g_n\big(\langle S, j\rangle\big)$ is applied. This transformation generates the message digest of the required length n from the internal state. First, the

Input: Internal state $\langle S, j \rangle$, 64-byte message block X.
Output: The updated internal state $\langle S, j \rangle$.

for $i = 0$ to 255 **do**
    $j \leftarrow j + S[i] + X[r(i)]$
    swap$(S[i], S[j])$
end for
return $\langle S, j \rangle$

Fig. 1. The compression function of RC4-Hash, $\mathcal{C}\big(\langle S, j \rangle, X\big)$. All arithmetic is done modulo 256.

Input: Internal state $\langle S, j \rangle$ after processing the entire padded message.
Output: The message digest H.

$S \leftarrow S^{\mathrm{IV}} \circ S$
// OWT (one way transformation)
$T_1 \leftarrow S$
for $i = 0$ to 511 **do**
    $j \leftarrow j + S[i]$
    swap$(S[i], S[j])$
end for
$T_2 \leftarrow S$
$S \leftarrow T_1 \circ T_2 \circ T_1$
// HBG (hash byte generation)
for $i = 0$ to n **do**
    $j \leftarrow j + S[i]$
    swap$(S[i], S[j])$
    $H[i] \leftarrow S[S[i] + S[j]]$
end for
return H

Fig. 2. The output transformation of RC4-Hash, $g_n\big(\langle S, j \rangle\big)$. All arithmetic is done modulo 256.

permutation S is composed with the initial value permutation S^{IV}. The resulting permutation is saved as T_1. Then, two blank iterations of the compression function $\mathcal{C}$, *i.e.*, using a zero message block, are applied, resulting in T_2. Finally, S is replaced by a composition of the two saved permutations, $T_1 \circ T_2 \circ T_1$, and the message digest is generated using an algorithm similar to RC4's pseudo-random byte generation.

Figure 2 shows the definition of the entire output transformation. In the original description of RC4-Hash [1], the output transformation was further partitioned into the algorithms OWT ("one way transformation") and HBG ("hash byte generation"). These correspond to lines 2-9 and 10-15 of the algorithm in Fig. 2, respectively.

3 Fixed Points of the Compression Function $\mathcal{C}$

In this section, we describe how to construct two distinct types of fixed points for a certain number of iterations of the RC4-Hash compression function $\mathcal{C}$. Each

of these constructions is based on one of two types of "partial state rotations", which are introduced in two lemmata, Lemma 1 and Lemma 3.

3.1 Fixed Points of Type I

Lemma 1 (Partial state rotations of type I). *Consider an internal state* $\langle S, 0\rangle$ *of RC4-Hash with* $S = \{s_0, s_1, \ldots, s_{255}\}$. *Denote by* $\langle S', j'\rangle$ *the internal state reached after applying the compression function* $\mathcal{C}$ *using the message block* $X = \{x, x, \ldots, x\}$ *with* $x = 1 - s_0 \bmod 256$*:*

$$\langle S', j'\rangle = \mathcal{C}\big(\langle S, j\rangle, X\big) \ . \tag{2}$$

Now, it holds that

$$j' = 0 \qquad \textit{and} \qquad S'[i] = \begin{cases} s_0 & i = 0 \\ s_{i+1} & 1 \le i < 255 \\ s_1 & i = 255 \end{cases} \ . \tag{3}$$

Proof. Denote by $\langle S^{(i)}, j^{(i)}\rangle$ the internal state of RC4-Hash after the i-th step of the compression function $\mathcal{C}$. First, we prove by induction that for every $i < 256$ it holds that

$$\begin{cases} j^{(i)} = i + 1 \bmod 256 \ , & \text{and} \\ S^{(i)}[i + 1 \bmod 256] = s_0 \ . \end{cases} \tag{4}$$

It is clear that this holds before the first step, *i.e.*, for $i = -1$, since $j^{(-1)} = 0$ and $S^{(-1)}[0] = S[0] = s_0$. Assume that the condition holds after step i ($i < 255$). Then, the update of the pointer j in the $(i + 1)$-th step is

$$\begin{aligned} j^{(i+1)} &= j^{(i)} + S^{(i)}[i + 1] + X[r(i + 1)] \bmod 256 \\ &= (i + 1) + s_0 + (1 - s_0) \bmod 256 \\ &= i + 2 \bmod 256 \ . \end{aligned} \tag{5}$$

Thus, $S^{(i+1)}$ is found by swapping the $(i + 1)$-th and $(i + 2)$-th element of $S^{(i)}$. Hence, $S^{(i+1)}[i + 2 \bmod 256] = S^{(i)}[i + 1 \bmod 256] = s_0$, *i.e.*, the condition also holds after step $i + 1$.

After 255 steps, all the elements of S have been circularly shifted over one position, *i.e.*, $S^{(254)} = \{s_1, s_2, \ldots, s_{255}, s_0\}$. In the final step, the first and the last element of $S^{(254)}$ are swapped since $j^{(255)} = 0$, resulting in

$$S^{(255)} = S' = \{s_0, s_2, s_3, \ldots, s_{254}, s_{255}, s_1\} \ . \tag{6}$$

From this, the lemma follows. □

Table 1 gives a detailed illustration of Lemma 1. The first column of this table gives the step number i, the second column gives the new value of the pointer j, computed in this step. The last column contains the array S after the step, where the elements that were just swapped are encircled.

Based on this first type of partial state rotations, it is straightforward to construct fixed points for 255 iterations of the compression function $\mathcal{C}$ as is shown in the next theorem.

Table 1. Partial state rotations of type I

step i	$j^{(i)}$	$S^{(i)}$								
	0	s_0	s_1	s_2	s_3	s_4	$\cdots$	s_{253}	s_{254}	s_{255}
0	1	$\boldsymbol{s_1}$	$\boldsymbol{s_0}$	s_2	s_3	s_4	$\cdots$	s_{253}	s_{254}	s_{255}
1	2	s_1	$\boldsymbol{s_2}$	$\boldsymbol{s_0}$	s_3	s_4	$\cdots$	s_{253}	s_{254}	s_{255}
2	3	s_1	s_2	$\boldsymbol{s_3}$	$\boldsymbol{s_0}$	s_4	$\cdots$	s_{253}	s_{254}	s_{255}
3	4	s_1	s_2	s_3	$\boldsymbol{s_4}$	$\boldsymbol{s_0}$	$\cdots$	s_{253}	s_{254}	s_{255}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$		$\vdots$	$\vdots$	$\vdots$
254	255	s_1	s_2	s_3	s_4	s_5	$\cdots$	s_{254}	$\boldsymbol{s_{255}}$	$\boldsymbol{s_0}$
255	0	$\boldsymbol{s_0}$	s_2	s_3	s_4	s_5	$\cdots$	s_{254}	s_{255}	$\boldsymbol{s_1}$

Theorem 1 (Fixed points of type I). *Consider an internal state* $\langle S, 0\rangle$ *of RC4-Hash with* $S = \{s_0, s_1, \ldots, s_{255}\}$*. After 255 iterations of the compression function* $\mathcal{C}$*, each using the same message block* $X = \{x, x, \ldots, x\}$ *with* $x = 1 - s_0 \bmod 256$*, the same state is reached:*

$$\langle S, 0\rangle = \mathcal{C}^{255}\big(\langle S, 0\rangle, X\big) \ . \tag{7}$$

Proof. The repeated application of Lemma 1 proves the theorem. □

Note that the only requirement for the construction of a fixed point of type I is that the pointer j has to be zero in the starting state. There are no conditions on the contents of the array S. Also, when given a suitable starting state, constructing a fixed point requires only a negligible amount of work, *i.e.*, one subtraction modulo 256 to compute the message byte $x = 1 - s_0 \bmod 256$.

3.2 Fixed Points of Type II

The message reordering $r(\cdot)$ has an interesting property which allows for another type of partial state rotations.

Lemma 2. *The message reordering* $r(\cdot)$ *does not reorder message bytes with an even index to odd-numbered positions, or vice versa. In other words,*

$$\forall i, 0 \le i < 256 : r(i) \equiv i \pmod 2 \ . \tag{8}$$

Proof. The lemma follows in a straightforward way from the definition of $r(\cdot)$ in Table 5. □

Lemma 3 (Partial state rotations of type II). *Consider an internal state* $\langle S, 1\rangle$ *of RC4-Hash with* $S = \{s_0, s_1, \ldots, s_{255}\}$*. Denote by* $\langle S', j'\rangle$ *the internal state reached after applying the compression function* $\mathcal{C}$ *using the message block* $X = \{x_0, x_1, x_0, x_1, \ldots, x_0, x_1\}$ *with* $x_0 = 1 - s_0 \bmod 256$ *and* $x_1 = 1 - s_1 \bmod 256$*:*

$$\langle S', j'\rangle = \mathcal{C}\big(\langle S, j\rangle, X\big) \ . \tag{9}$$

Now, it holds that

$$j' = 1 \qquad \textit{and} \qquad S'[i] = \begin{cases} s_i & 0 \le i < 2 \\ s_{i+2} & 2 \le i < 254 \\ s_{i-252} & 254 \le i < 256 \end{cases} . \tag{10}$$

Proof. Denote by $\langle S^{(i)}, j^{(i)} \rangle$ the internal state of RC4-Hash after the i-th step of the compression function $\mathcal{C}$. Note that, because of Lemma 2 and the definition of X, $X[r(i)] = x_{i \bmod 2} = 1 - s_{i \bmod 2}$. First, we prove by induction that for every $i < 256$ it holds that

$$\begin{cases} j^{(i)} = i + 2 \bmod 256 \ , & \text{and} \\ S^{(i)}[i + 1 \bmod 256] = s_{i+1 \bmod 2} \ , & \text{and} \\ S^{(i)}[i + 2 \bmod 256] = s_{i \bmod 2} \ . \end{cases} \tag{11}$$

It is clear that this holds before the first step, *i.e.*, for $i = -1$, since $j^{(-1)} = 1$, $S^{(-1)}[0] = S[0] = s_0$ and $S^{(-1)}[1] = S[1] = s_1$. Assume that the condition holds after step i ($i < 255$). Then, the update of the pointer j in the $(i+1)$-th step is

$$\begin{aligned} j^{(i+1)} &= j^{(i)} + S^{(i)}[i+1] + X[r(i+1)] \bmod 256 \\ &= (i+2) + s_{i+1 \bmod 2} + (1 - s_{i+1 \bmod 2}) \bmod 256 \\ &= i + 3 \bmod 256 \ . \end{aligned} \tag{12}$$

Thus, $S^{(i+1)}$ is found by swapping the $(i+1)$-th and $(i+3)$-th element of $S^{(i)}$. Hence, $S^{(i+1)}[i + 3 \bmod 256] = S^{(i)}[i + 1 \bmod 256] = s_{i+1 \bmod 2}$. Of course, $S^{(i+1)}[i + 2 \bmod 256] = S^{(i)}[i + 2 \bmod 256] = s_{i \bmod 2}$. This implies that the condition also holds for step $i + 1$.

After 254 steps, all the elements of S have been circularly shifted over two position, *i.e.*, $S^{(253)} = \{s_2, s_3, s_4, \ldots, s_{255}, s_0, s_1\}$. Since $j^{(254)} = 0$ and $j^{(255)} = 1$, the swaps made in the last two steps result in the following state

$$S^{(255)} = S' = \{s_0, s_1, s_4, \ldots, s_{255}, s_2, s_3\} \ . \tag{13}$$

From this, the lemma follows. □

Table 2 gives a detailed illustration of Lemma 3. The notations are the same as in Table 1. Based on this type of partial state rotations, fixed points for 127 iterations of the compression function $\mathcal{C}$ can be constructed, as is shown in the next theorem.

Theorem 2 (Fixed points of type II). *Consider an internal state $\langle S, 1 \rangle$ of RC4-Hash with $S = \{s_0, s_1, \ldots, s_{255}\}$. After 127 iterations of the compression function $\mathcal{C}$, each using the same message block $X = \{x_0, x_1, x_0, x_1, \ldots, x_0, x_1\}$ with $x_0 = 1 - s_0 \bmod 256$ and $x_1 = 1 - s_1 \bmod 256$, the same state is reached:*

$$\langle S, 1 \rangle = \mathcal{C}^{127}\big(\langle S, 1 \rangle, X \big) \ . \tag{14}$$

Proof. The repeated application of Lemma 3 proves the theorem. □

Table 2. Partial state rotations of type II

step i	$j^{(i)}$	$S^{(i)}$								
	1	s_0	s_1	s_2	s_3	s_4	$\cdots$	s_{253}	s_{254}	s_{255}
0	2	$\boldsymbol{s_2}$	s_1	$\boldsymbol{s_0}$	s_3	s_4	$\cdots$	s_{253}	s_{254}	s_{255}
1	3	s_2	$\boldsymbol{s_3}$	s_0	$\boldsymbol{s_1}$	s_4	$\cdots$	s_{253}	s_{254}	s_{255}
2	4	s_2	s_3	$\boldsymbol{s_4}$	s_1	$\boldsymbol{s_0}$	$\cdots$	s_{253}	s_{254}	s_{255}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$		$\vdots$	$\vdots$	$\vdots$
253	255	s_2	s_3	s_4	s_5	s_6	$\cdots$	$\boldsymbol{s_{255}}$	s_0	$\boldsymbol{s_1}$
254	0	$\boldsymbol{s_0}$	s_3	s_4	s_5	s_6	$\cdots$	s_{255}	$\boldsymbol{s_2}$	s_1
255	1	s_0	$\boldsymbol{s_1}$	s_4	s_5	s_6	$\cdots$	s_{255}	s_2	$\boldsymbol{s_3}$

Note that, as for fixed points of type I, the only requirement for the construction of a fixed point of type II is that the j pointer has a certain value in the starting state. There are no conditions on the contents of the array S. Constructing a fixed point of type II, when given a suitable starting state, also requires only a negligible amount of work, *i.e.*, two subtractions modulo 256 to compute the message bytes $x_0 = 1 - s_0 \bmod 256$ and $x_1 = 1 - s_1 \bmod 256$.

One could try to further generalise this to longer cyclic patterns. However, the message byte reordering $r(\cdot)$ prevents this as there is no $p > 2$ for which it holds that

$$\forall i, 0 \leq i < 256 : r(i) \equiv i \pmod{p} \ . \tag{15}$$

3.3 Relation to Finney States

A Finney state [4] is an RC4-state where $j = i + 1$ and $S[i] = 1$. From the definition of the RC4 stream cipher, see Fig. 5, it follows that if the current state is a Finney state, the next state must also be a Finney state. Similarly, a Finney state can only arise from a Finney state. In a Finney state, the element "1" is simply moved to the next position in the array S and j is incremented. The initialisation of the RC4 pseudo-random byte generator, see Fig. 5, ensures that the initial state is not a Finney state. Hence, Finney states can never occur in RC4.

In RC4-Hash, however, we can achieve a similar pattern. This is exactly what is done in the case of partial state rotations of type I. The extra freedom coming from the message input is exploited to ensure that the element $S[i]$ is always moved to the next position, such that it is again used to update j in the next iteration. Partial state rotations of type II are a generalisation of this, using two elements in an alternating way.

4 Collisions for RC4-Hash

This section describes how to use fixed points for a number of iterations of the compression function $\mathcal{C}$ to construct colliding message pairs for RC4-Hash. In

order to be able to construct fixed points, the value of the pointer j in the internal state of RC4-Hash has to be equal to zero (for fixed points of type I) or one (for fixed points of type II), as described in Sect. 3. Although the initial value of j is zero, we cannot make use of the first block because we do not have control over its first byte, which contains the digest length.

Consider fixed points of type I, *i.e.*, we want $j = 0$. Since j can only take 2^8 possible values, we can simply search for a prefix block P which leads to a suitable internal state:

$$\langle S_0, 0\rangle = \mathcal{C}\big(\langle S^{\mathrm{IV}}, 0\rangle, \mathrm{bin}_8(n)||P\big) \ . \tag{16}$$

We expect to find a suitable prefix block after about 2^8 random trials. At this point, we can easily construct a fixed point for this state $\langle S_0, 0\rangle$ by applying Theorem 1. Denote by $M^{0,0}$ the message block that is used 255 times in this fixed point.

Then, we search for an additional message block $M^{0,1}$ which transforms the state $\langle S_0, 0\rangle$ into $\langle S_1, 0\rangle$:

$$\langle S_1, 0\rangle = \mathcal{C}\big(\langle S_0, 0\rangle, M^{0,1}\big) \ . \tag{17}$$

Again, the only condition on $M^{0,1}$ is that the value of the j pointer is not changed by the compression function $\mathcal{C}$. The expected number of random trials required to find a suitable message block is again about 2^8. For the state $\langle S_1, 0\rangle$, it is also possible to construct a fixed point of type I, using Theorem 1. Denote the message block used in this fixed point by $M^{1,1}$. Now, consider the following two messages:

$$\begin{aligned} M &= P||M^{0,1}||\overbrace{M^{1,1}||\cdots||M^{1,1}}^{255} \ , \\ M^\star &= P||\underbrace{M^{0,0}||\cdots||M^{0,0}}_{255}||M^{0,1} \ . \end{aligned} \tag{18}$$

As shown in Fig. 3, these messages form a collision. Indeed, after processing the 257-th block, the internal state of RC4-Hash is $\langle S_1, 0\rangle$ for both messages,

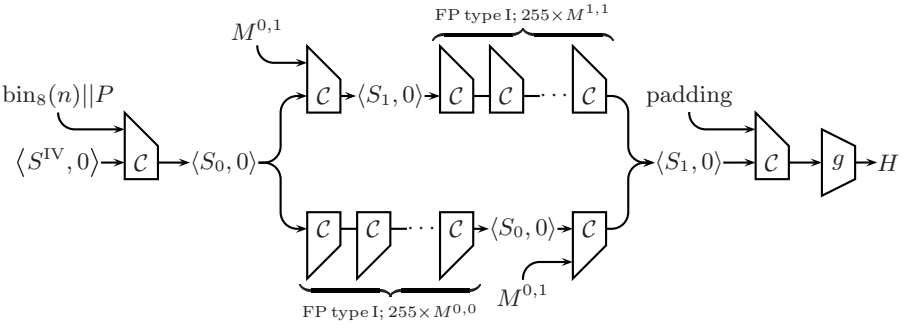


Fig. 3. A collision pair for RC4-Hash using fixed points of type I

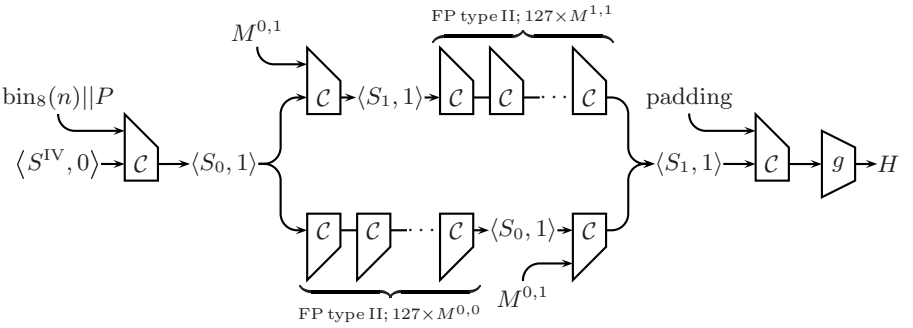


Fig. 4. A collision pair for RC4-Hash using fixed points of type II

i.e., an internal state collision is reached. The extra padding block containing the message length and the output transformation maintain the collision. The expected total time complexity is only 2^9 evaluations of the compression function $\mathcal{C}$. Note that verifying the collision requires about the same effort, since hashing M and $M^\star$ requires two times 258 calls to the compression function $\mathcal{C}$.

Using fixed points of type II, collisions can be found in a completely similar way, as Fig. 4 illustrates. The only differences are that we now require $j = 1$, and that the fixed points only contain 127 iterations of the compression function $\mathcal{C}$. The expected time complexity is also 2^9. If we do not fix in advance which type of fixed points to use, but let this depend on which kind of prefix block is found first, the expected time complexity can be lowered slightly to $2^7 + 2^8$ compression function evaluations.

There is no need to restrict the prefix block P or the message block $M^{0,1}$ to be only a single block. Using multiple blocks does not (significantly) increase the

Table 3. Example collision pair for RC4-Hash$_{64}$, using fixed points of type I

	M	$M^\star$
block 1 (63 bytes)	s.␣IndestEEGE␣AnD␣B.␣pReNeEl␣-␣ cosIc␣-␣cOlLisIoNS␣FoR␣rC4-Hash.	
block 2 (64 bytes)	thiS␣MEssAgE␣Is␣pArT␣oF␣a␣colLis ion␣EXaMpLe␣for␣RC4-HASH.␣COSIC.	AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
blocks 3–256 (254 × 64 bytes)	aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa ⋮ aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa	AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA ⋮ AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
block 257 (64 bytes)	aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa	thiS␣MEssAgE␣Is␣pArT␣oF␣a␣colLis ion␣EXaMpLe␣for␣RC4-HASH.␣COSIC.
RC4-Hash$_{64}(M)$ = RC4-Hash$_{64}(M^\star)$	0093b4baefdc64f93d7081978808c49d1286523696e6d4a35ab64f1e42695aff 79ce81eae91cb47673c4989238fab010f47466906fa65bed88753802c71ae82b$_x$	

Table 4. Example collision pair for RC4-Hash$_{64}$, using fixed points of type II

	M	$M^\star$
block 1 (63 bytes)	s.␣IndesTeEGE␣ANd␣b.␣pREneEl␣-␣ cosIc␣-␣colLISioNS␣For␣Rc4-hAsH.	
block 2 (64 bytes)	thiS␣MesSagE␣IS␣pArT␣of␣a␣collis ioN␣EXAmPle␣FOr␣rc4-HASH.␣COSIC.	aBaBaBaBaBaBaBaBaBaBaBaBaBaBaB aBaBaBaBaBaBaBaBaBaBaBaBaBaBaB
blocks 3–128 (126 × 64 bytes)	abababababababababababababab abababababababababababababab ⋮ abababababababababababababab	aBaBaBaBaBaBaBaBaBaBaBaBaBaBaB aBaBaBaBaBaBaBaBaBaBaBaBaBaBaB ⋮ aBaBaBaBaBaBaBaBaBaBaBaBaBaBaB
block 129 (64 bytes)	abababababababababababababab abababababababababababababab	thiS␣MesSagE␣IS␣pArT␣of␣a␣collis ioN␣EXAmPle␣FOr␣rc4-HASH.␣COSIC.
RC4-Hash$_{64}(M)$ = RC4-Hash$_{64}(M^\star)$	0023dd337650ef0d9b5e77be533ea644198ff0d8f1d8190628d95b9dd04dadf5 d9cd2c1ad8adc8555f03ea3819df4128bc96462a53c7e0cc1afffe78db3bd652$_x$	

expected time complexity for finding a collision pair, if only the last block of P, resp. $M^{0,1}$, is varied in order to obtain the desired value for the pointer j. Of course, a colliding message pair can always be extended with an equal suffix.

Tables 3 and 4 give examples of colliding message pairs for RC4-Hash$_{64}$, constructed using fixed points of type I and type II, respectively. Additional constraints were imposed to arrive at meaningful messages.

5 Discussion

Kelsey-Schneier Second Preimages. Since fixed points of the compression function of RC4-Hash can be constructed very easily, one may consider to use them to mount a Kelsey-Schneier second preimage attack [6]. This involves building expandable messages, *i.e.*, messages of varying length, which all collide on the intermediate hash result immediately after processing the message. The main problem which makes the Kelsey-Schneier second preimage attack fail for RC4-Hash, is the very large internal state of RC4-Hash. Because of this, the Kelsey-Schneier attack is much slower than exhaustive search in this case.

Multicollisions. A multicollision is a (large) set of messages that all hash to the same value. Multicollisions and their applications were described by Joux [5], although Coppersmith already used them in 1985 [2]. In order to obtain multicollisions for RC4-Hash, we simply concatenate the method from Sect. 4 several times. Concatenating it k times yields 2^k colliding messages. Actually, only part of the method needs to be repeated k times. Indeed, as the value of the pointer j is maintained by the fixed points, only the search for message blocks $M^{0,1}$ has to be repeated. Thus, the expected time for finding 2^k colliding messages for RC4-Hash is $2^7 + k \cdot 2^8$ compression function evaluations. Naturally, also the method of Kelsey and Schneier [6] to construct multicollisions can be applied, and both methods can even be combined.

Mitigating the Attack. The collision attack described in this paper is built on the existence of two types of fixed points of the compression function of RC4-Hash, which were described in Sect 3. These fixed points use patterns where all the (reordered) message bytes are equal (type I) or alternate between two values (type II). Replacing the message reordering $r(\cdot)$ with a message expansion that guarantees that such patterns can never occur foils the attack. Another approach would be to introduce asymmetry, for instance using intermediate rounds.

6 Conclusion

We have shown that RC4-Hash is not collision resistant. There exist two distinct types of fixed points for a number of iterations of the RC4-Hash compression function $\mathcal{C}$. These can be used to construct colliding message pairs with an expected effort of less than 2^9 compression function evaluations. This also leads to multicollisions, yielding 2^k colliding messages with an expected effort of $2^7 + k \cdot 2^8$ compression function evaluations.

References

1. Chang, D., Gupta, K.C., Nandi, M.: "RC4-Hash: A New Hash Function Based on RC4". In: Barua, R., Lange, T. (eds.) INDOCRYPT 2006. LNCS, vol. 4329, pp. 80–94. Springer, Heidelberg (2006)
2. Coppersmith, D.: Another Birthday Attack. In: Williams, H.C. (ed.) CRYPTO 1985. LNCS, vol. 218, pp. 14–17. Springer, Heidelberg (1986)
3. De Cannière, C., Rechberger, C.: Finding SHA-1 Characteristics: General Results and Applications. In: Lai, X., Chen, K. (eds.) ASIACRYPT 2006. LNCS, vol. 4284, pp. 1–20. Springer, Heidelberg (2006)
4. Finney, H.: An RC4 cycle that can't happen, Newsgroup post in sci. crypt (September 1994)
5. Joux, A.: Multicollisions in Iterated Hash Functions. Application to Cascaded Constructions. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 306–316. Springer, Heidelberg (2004)
6. Kelsey, J., Schneier, B.: Second Preimages on n-Bit Hash Functions for Much Less than 2^n Work. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 474–490. Springer, Heidelberg (2005)
7. Lucks, S.: A Failure-Friendly Design Principle for Hash Functions. In: Roy, B. (ed.) ASIACRYPT 2005. LNCS, vol. 3788, pp. 474–494. Springer, Heidelberg (2005)
8. Schneier, B.: Applied Cryptography, 2nd edn. John Wiley & Sons, Chichester (1996)
9. Wang, X., Yu, H.: How to Break MD5 and Other Hash Functions. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 19–35. Springer, Heidelberg (2005)
10. Wang, X., Yu, H., Yin, Y.L.: Efficient Collision Search Attacks on SHA-0. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 1–16. Springer, Heidelberg (2005)
11. Wang, X., Yin, Y.L., Yu, H.: Finding Collisions in the Full SHA-1. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 17–36. Springer, Heidelberg (2005)

Appendix

Table 5. The message reordering $r(\cdot)$

0,	1,	2,	3,	4,	5,	6,	7,	8,	9,	10,	11,	12,	13,	14,	15,	16,	17,	18,	19,	20,	21,
22,	23,	24,	25,	26,	27,	28,	29,	30,	31,	32,	33,	34,	35,	36,	37,	38,	39,	40,	41,	42,	43,
44,	45,	46,	47,	48,	49,	50,	51,	52,	53,	54,	55,	56,	57,	58,	59,	60,	61,	62,	63,		
0,	55,	46,	37,	28,	19,	10,	1,	56,	47,	38,	29,	20,	11,	2,	57,	48,	39,	30,	21,	12,	3,
58,	49,	40,	31,	22,	13,	4,	59,	50,	41,	32,	23,	14,	5,	60,	51,	42,	33,	24,	15,	6,	61,
52,	43,	34,	25,	16,	7,	62,	53,	44,	35,	26,	17,	8,	63,	54,	45,	36,	27,	18,	9,		
0,	57,	50,	43,	36,	29,	22,	15,	8,	1,	58,	51,	44,	37,	30,	23,	16,	9,	2,	59,	52,	45,
38,	31,	24,	17,	10,	3,	60,	53,	46,	39,	32,	25,	18,	11,	4,	61,	54,	47,	40,	33,	26,	19,
12,	5,	62,	55,	48,	41,	34,	27,	20,	13,	6,	63,	56,	49,	42,	35,	28,	21,	14,	7,		
0,	47,	30,	13,	60,	43,	26,	9,	56,	39,	22,	5,	52,	35,	18,	1,	48,	31,	14,	61,	44,	27,
10,	57,	40,	23,	6,	53,	36,	19,	2,	49,	32,	15,	62,	45,	28,	11,	58,	41,	24,	7,	54,	37,
20,	3,	50,	33,	16,	63,	46,	29,	12,	59,	42,	25,	8,	55,	38,	21,	4,	51,	34,	17.		

Table 6. The initial value permutation S^{IV}

145,	57,	133,	33,	65,	49,	83,	61,	113,	171,	63,	155,	74,	50,	132,	248,
236,	218,	192,	217,	23,	36,	79,	72,	53,	210,	38,	59,	54,	208,	185,	12,
233,	189,	159,	169,	240,	156,	184,	200,	209,	173,	20,	252,	96,	211,	143,	101,
44,	223,	118,	1,	232,	35,	239,	9,	114,	109,	161,	183,	88,	66,	219,	78,
157,	174,	187,	193,	199,	99,	52,	120,	89,	166,	18,	76,	241,	13,	225,	6,
146,	151,	207,	177,	103,	45,	148,	32,	29,	234,	7,	16,	19,	91,	108,	186,
116,	62,	203,	158,	180,	149,	67,	105,	247,	3,	128,	215,	121,	127,	179,	175,
251,	104,	246,	98,	140,	11,	134,	221,	24,	69,	190,	154,	253,	168,	68,	230,
58,	153,	188,	224,	100,	129,	124,	162,	15,	117,	231,	150,	237,	64,	22,	152,
165,	235,	227,	139,	201,	84,	213,	77,	80,	197,	250,	126,	202,	39,	0,	94,
42,	243,	228,	87,	82,	27,	141,	60,	160,	46,	125,	112,	181,	242,	167,	92,
198,	172,	170,	55,	115,	30,	107,	17,	56,	31,	135,	229,	40,	111,	37,	222,
182,	25,	43,	119,	244,	191,	122,	102,	21,	93,	97,	131,	164,	10,	130,	47,
176,	238,	212,	144,	41,	14,	249,	220,	34,	136,	71,	48,	142,	73,	123,	204,
206,	4,	216,	196,	214,	137,	255,	195,	26,	8,	51,	178,	2,	138,	254,	90,
194,	81,	245,	106,	95,	75,	86,	163,	205,	70,	226,	28,	147,	85,	5,	110.

Input: Key K consisting of κ bytes.
Output: Initial internal state of RC4, $\langle S, i, j\rangle$.
1: *// RC4 Key Scheduling Algorithm (KSA)*
2: $S \leftarrow \{0, 1, \cdots, 255\}$
3: $j \leftarrow 0$
4: **for** $i = 0$ to 255 **do**
5: $\quad j \leftarrow j + S[i] + K[i \bmod \kappa]$
6: $\quad \mathrm{swap}(S[i], S[j])$
7: **end for**
8: **return** $\langle S, 0, 0\rangle$

Input: RC4 internal state $\langle S, i, j\rangle$.
Output: One byte of keystream, updated internal state.
1: *// RC4 pseudo-random byte generation (PRBG)*
2: $i \leftarrow i + 1$
3: $j \leftarrow j + S[i]$
4: $\mathrm{swap}(S[i], S[j])$
5: **return** $S[S[i] + S[j]]$

Fig. 5. The RC4 stream cipher, consisting of a key scheduling algorithm (top) and a pseudo-random byte generator (bottom). All arithmetic is done modulo 256 [8].

New Applications of Differential Bounds of the SDS Structure

Jiali Choy and Khoongming Khoo

DSO National Laboratories
20 Science Park Drive, Singapore 118230
{cjiali,kkhoongm@}dso.org.sg

Abstract. In this paper, we present some new applications of the bounds for the differential probability of a SDS (Substitution-Diffusion-Substitution) structure by Park et al. at FSE 2003. Park et al. have applied their result on the AES cipher which uses the SDS structure based on MDS matrices. We shall apply their result to practical ciphers that use SDS structures based on $\{0,1\}$-matrices of size $n \times n$. These structures are useful because they can be efficiently implemented in hardware. We prove a bound on $\{0,1\}$-matrices to show that they cannot be MDS and are almost-MDS only when $n = 2, 3$, or 4. Thus we have to apply Park's result whenever $\{0,1\}$-matrices where $n \geq 5$ are used because previous results only hold for MDS and almost-MDS diffusion matrices. Based on our bound, we also show that the $\{0,1\}$-matrices used in E2, Camellia, and MCrypton are optimal or almost-optimal among $\{0,1\}$-matrices. Using Park's result, we prove differential bounds for the E2 and MCrypton ciphers, from which we can deduce their security against boomerang attack and some of its variants. At ICCSA 2006, Khoo and Heng constructed block cipher-based universal hash functions, from which they derived Message Authentication Codes (MACs) which are faster than CBC-MAC. Park's result provides us with the means to obtain a more accurate bound for their universal hash function. With this bound, we can restrict the number of MAC's performed before a change of MAC key is needed.

Keywords: SPN, branch number, differential, $\{0,1\}$-matrices, universal hash functions.

1 Introduction

Differential cryptanalysis is one of the most well-known attacks on block ciphers. It exploits differential characteristics, which consist of a sequence of difference patterns in consecutive rounds, with high probability. However, even if the maximal characteristic probability is low, one cannot conclude that the cipher is secure against differential attack as it may not be necessary to fix the values of the input and output differences for intermediate rounds to perform the attack. Instead, one must turn to the concept of a *differential*, which is the set of all differential characteristics with the same initial and terminal difference. To be

T.-C. Wu et al. (Eds.): ISC 2008, LNCS 5222, pp. 367–384, 2008.

provably secure against differential cryptanalysis, the differential probability of all differentials must be low enough.

Another motivation for studying differential probability is to analyze a cipher's security against boomerang attacks [21] and its variants such as amplified boomerang attacks [13] and rectangle attacks [3]. In usual differential cryptanalysis, it is not easy to determine the differential probability of a cipher if it has too many rounds. In boomerang-based attacks, a cipher is split into two shorter subciphers from which it is easier to find a differential with high probability for each half. These differentials are then combined to form boomerang attacks based on adaptive chosen plaintext-ciphertexts, or amplified boomerang and rectangle attacks based on chosen plaintexts. If we can prove that a cipher has low differential probability over reduced rounds, then we can prove security against these attacks.

S. Hong et al. [11] analyzed the provable security of the SPN (Substitution-Permutation Network) structure depicted in Figure 1 in Appendix C. This structure is widely used in many block cipher designs as it is highly parallelizable and its security is more easily analyzed. Each round consists of key addition, substitution, and permutation of bits. The diffusion layer is paramount to the whole design as it provides the avalanche effect to ensure good randomization. An SPN cipher with a low branch number associated with its diffusion layer is regarded as weak against differential and linear cryptanalysis. In particular, their paper dealt with the provable security against differential and linear cryptanalysis of an SPN structure with a maximal distance separable (MDS) diffusion layer and an almost-MDS diffusion layer.

In [18], Park et al. extended Hong's results in two directions:

(i) Improvement 1: They took into account the differential and linear probability distribution of the S-boxes involved as compared to Hong et al. who considered the maximal differential and linear probability of the S-box. This enabled them to derive differential and linear probability bounds which are better (lower) than previously known bounds in [8] and [18].
(ii) Improvement 2: They derived the differential and linear probability of the SDS (Substitution-Diffusion-Substitution) structure for diffusion layers with any specified differential or linear branch number respectively, as opposed to Hong et al.'s results which are only applicable for MDS and almost-MDS diffusion layers.

They then went on to prove differential and linear bounds for the AES cipher which are better than the known bounds by Rijmen and Daemen in [8]. This demonstrates the advantage of their result for Improvement 1. However, the second advantage of their analysis is that we can derive the differential and linear probability of SDS structures where the diffusion layer is not MDS or almost-MDS (Improvement 2 above). We shall demonstrate the practicability of their results by applying these results to SDS structures based on diffusion layers which are $\{0, 1\}$-matrices. They are widely used in ciphers like Camellia, E2 and MCrypton [1,12,15] because they require less gates when implemented in hardware and will be well-suited to constrained environments such as RFID tags.

First, we prove an upper bound for the branch numbers of $\{0,1\}$-matrices of size $n \times n$. This will provide us with an idea of $\{0,1\}$-matrices which are optimal or almost-optimal. We show that $\{0,1\}$-matrices are never MDS and they are almost-MDS only when $n = 2, 3$, or 4. Thus we need to apply Park's result whenever $n \geq 5$. The ciphers E2, Camellia, and MCrypton use $\{0,1\}$-matrices. With our bounds, we can show that the diffusion mappings in these ciphers are optimal or almost-optimal among $\{0,1\}$-matrices.

Second, we apply Park's result [18] to derive a general formula for the differential probability of the SDS structure which uses an affine transform of the inverse S-box over $GF(2^m)$ and a diffusion mapping with any arbitrary branch number. This will allow us to prove the differential probability of E2 [12] and MCrypton [15]. Furthermore, we are also able to prove the resistance of these ciphers against boomerang attack [21] and some of its variants [3,13].

Third, we improve on an upper bound for a universal hash construction by Khoo and Heng [14]. In their paper, the authors showed that we can implement a block cipher-based universal hash function which uses reduced rounds and is parallelizable. This results in a universal hash function-based message authentication code (MAC) which is faster than CBC-MAC. However, we show that the upper bound in [14] is approximate and is, in fact, higher than the actual bound. By applying on Park's result [18], we can give a more accurate bound for the MAC. Based on this bound, we can restrict the number of authentications performed before a change of MAC-key is needed.

2 Definitions

A measure of the efficiency of block ciphers against differential cryptanalysis is to have low maximal differential.

Definition 1. *The* maximal differential *of a function* $f : GF(2)^w \rightarrow GF(2)^w$ *is defined as*

$$\Delta_f = \underset{\Delta x \neq 0, \Delta y}{max} | \{x \in GF(2)^w | f(x) \oplus f(x \oplus \Delta x) = \Delta y\} |$$

In the subsequent sections, we consider an SPN structure with an mn-bit round function. Let each S-box S_i be an $m \times m$ bijective function

$$S_i : GF(2)^m \rightarrow GF(2)^m \qquad (1 \leq i \leq n).$$

Also we assume that the round keys, which are XORed with the input data at each round, are independent and uniformly random.

Definition 2. *For any given* $\Delta x, \Delta y, \Gamma x, \Gamma y \in GF(2)^m$*, the* differential probability *of each* S_i *is defined as*

$$DP^{S_i}(\Delta x \rightarrow \Delta y) = \frac{\#\{x \in GF(2)^m | S_i(x) \oplus S_i(x \oplus \Delta x) = \Delta y\}}{2^m},$$

where we consider Δx *to be the input difference and* Δy *the output difference. The* maximal differential probability *of* S_i *is defined as*

$$DP((S_i)_{max}) = \underset{\Delta x \neq 0, \Delta y}{max} DP^{S_i}(\Delta x \rightarrow \Delta y).$$

The maximal values *of* $DP((S_i)_{max})$ *for* $1 \leq i \leq n$ *is defined as*

$$p = \underset{1 \leq i \leq n}{max} (DP(S_i)_{max}).$$

S_i is strong against differential cryptanalysis if $DP((S_i)_{max})$ is low enough, while a substitution layer is strong if p is low enough. However, it is important to note that a strong S-box and a strong substitution layer do not ensure a secure SPN structure against differential attacks. To evaluate provable security, one has to take the diffusion layer into account as well. The latter is an invertible linear mapping, the purpose of which is to provide an avalanche effect, both in the context of differential and linear approximations. In the differential context, this means that small input changes should cause large output changes, and conversely, to produce a small output change, a large input change should be necessary.

A *differentially active* S-box is an S-box given a non-zero input difference. Differentially inactive S-boxes with zero input difference always have zero output difference with probability 1. Due to the independent round key assumption, the key addition layer in Figure 1 has no effect on the number of active S-boxes.

Definition 3. *Let* $x = (x_1, \cdots, x_n)^t \in GF(2^m)^n$. *The* Hamming weight *of* x *is defined as*

$$wt(x) = \#\{i | x_i \neq 0\},$$

which is the number of non-zero m*-bit characters in* x.

Now define a SDS (Substitution-Diffusion-Substitution) function as shown in Figure 2 in Appendix C. Let the diffusion layer of the SDS function be denoted by D, its input difference by $\Delta x = x \oplus x^*$, its output difference by $\Delta y = y \oplus y^* = D(x) \oplus D(x^*)$. If D is linear, we have $\Delta y = D(\Delta x)$. The minimum number of differentially active S-boxes of the SDS function is given by the branch number of the diffusion layer.

Definition 4. *The* branch number *of a diffusion layer* D *is defined as:*

$$Br(D) = \min_{v \neq 0} \{wt(v) + wt(D(v))\} \tag{1}$$

If we want to find the number of active S-boxes in two consecutive rounds of the SPN structure, we may disregard the two key addition layers since they have no influence on the number. Consequently, we only need to consider the SDS function. Therefore, $Br(D)$ gives a measure of the worst case diffusion: it is a lower bound for the number of active S-boxes in two consecutive rounds

of a differential characteristic approximation. Since a cryptanalyst will always exploit the worst case, this is a good measure of the diffusion property.

Definition 5. *A diffusion layer is* maximal distance separable (MDS) *if* $Br(D)$ *is* $n+1$*; it is called* almost-MDS *if* $Br(D)$ *is equal to* n*.*

Proposition 1. *([11, Theorem 1,Theorem 3]) Assume that the round keys, which are XORed to the input data at each round, are independent and uniformly random. The probability of each differential of the SDS structure (and consequently, the SPN structure)*

(i) is bounded by p^n *if* D *is MDS, i.e. if* $Br(D) = n+1$*;*
(ii) is bounded by p^{n-1} *if* D *is almost-MDS, i.e. if* $Br(D) = n$*.*

However, the above lemma has been improved by Park et al. to apply to SDS structures where the diffusion layer need not be MDS or almost-MDS.

Proposition 2. *([18, Theorem 1]) Assume that the round keys, which are XORed to the input data at each round, are independent and uniformly random. If* $Br(D) = k$*, the probability of each differential of the SDS structure (and consequently, the SPN structure) is bounded by:*

$$\max\left(\max_{1\le i\le n}\max_{1\le u\le 2^m-1}\sum_{j=1}^{2^m-1} DP^{S_i}(u\to j)^k, \max_{1\le i\le n}\max_{1\le u\le 2^m-1}\sum_{j=1}^{2^m-1} DP^{S_i}(j\to u)^k\right)$$

As a corollary, Park et al. obtained the following result which can be viewed as a direct generalization of Hong et al.'s result.

Proposition 3. *([18, Corollary 1]) Assume that the round keys, which are XORed to the input data at each round, are independent and uniformly random. The probability of each differential of the SDS structure (and consequently, the SPN structure) is bounded by* p^{k-1} *if* $Br(D) = k$*.*

3 Branch Number of $\{0,1\}$-Matrices

We shall look at the differential probability of the SDS structure where the diffusion layer is a matrix with entries 0 or 1, which we call $\{0,1\}$-matrices. The reason we study $\{0,1\}$-matrices is because they are faster to compute than MDS transforms which are used in many block ciphers like Rijndael, Square and Shark [8,10,19]. Another reason is that in hardware implementation, they will take up less space and thus allow for more compact implementation.

In this section, we consider a closely related problem: The study of the branch number of such matrices. The proofs of results in this section can be found in Appendix A.1, A.2, and A.3.

Theorem 1. *Let $A : GF(2^m)^n \rightarrow GF(2^m)^n$ be an $n \times n$ $\{0,1\}$-matrix over $GF(2^m)$. Then the branch number of A is at most $\frac{2n+4}{3}$.*

By studying the upper bound of Theorem 1, it is easy to deduce the following Corollary.

Corollary 1. *Let $A : GF(2^m)^n \rightarrow GF(2^m)^n$ be an $n \times n$ $\{0,1\}$-matrix over $GF(2^w)$. Then A is not a MDS matrix and it can be an almost-MDS matrix only when $n = 2, 3,$ or 4.*

In Table 1 in Appendix B, we list the upper bounds for the branch number of $\{0,1\}$-matrices for different n. We see from Corollary 1 that when we want to deduce the true differential probability of SDS structures, where the diffusion layer is represented by a $\{0,1\}$-matrix, we can only apply the known results (on almost-MDS matrices) from [11] for $n = 2, 3$, or 4. For $n \geq 5$, we have to apply Theorem 1 from [18].

3.1 Some $\{0,1\}$-Matrices with Optimal Branch Numbers

Based on Theorem 1, we give the following definition.

Definition 6. *A $\{0,1\}$-matrix A of size $n \times n$ is called* optimal *(w.r.t. Theorem 1) if its branch number is $\lfloor \frac{2n+4}{3} \rfloor$. It is called* almost-optimal *(w.r.t. Theorem 1) if its branch number is $\lfloor \frac{2n+4}{3} \rfloor - 1$.*

We shall look at some $\{0,1\}$-matrices with optimal or almost optimal branch numbers. The first matrix we shall study has 0 on the diagonal and 1 elsewhere.

Proposition 4

(i) Consider the following $n \times n$ matrix A, $n \geq 2$, which maps from $GF(2^m)^n \rightarrow GF(2^m)^n$.

$$A = \begin{pmatrix} 0\,1\,1\,\ldots\,1 \\ 1\,0\,1\,\ldots\,1 \\ 1\,1\,0\,\ldots\,1 \\ \vdots\,\vdots\,\vdots\,\ddots\,\vdots \\ 1\,1\,1\,\ldots\,0 \end{pmatrix}. \tag{2}$$

The branch number of A is $\min(n, 4)$.

(ii) More generally, a $n \times n$ matrix $A : GF(2^m)^n \rightarrow GF(2^m)^n$ where each row and each column has 1 occurrence of 0 and $n-1$ occurrences of 1, has branch number $\min(n, 4)$.

(iii) When $n = 4$, the matrices in part (ii) are the only 4×4 $\{0,1\}$-matrices over $GF(2^m)$ with optimal branch number 4.

By referring to Table 1, we see that the matrices in Proposition 4 part (ii) are optimal among $\{0,1\}$-matrices when $n = 2, 3, 4, 5$ and almost-optimal when $n = 6$. These matrices are used in the MCrypton cipher where $n = 4$.

The following $\{0,1\}$-matrix acting on 8 bytes is used in the E2 and Camellia cipher:

$$A = \begin{pmatrix} 0\,1\,1\,1\,1\,1\,1\,0 \\ 1\,0\,1\,1\,0\,1\,1\,1 \\ 1\,1\,0\,1\,1\,0\,1\,1 \\ 1\,1\,1\,0\,1\,1\,0\,1 \\ 1\,1\,0\,1\,1\,1\,0\,0 \\ 1\,1\,1\,0\,0\,1\,1\,0 \\ 0\,1\,1\,1\,0\,0\,1\,1 \\ 1\,0\,1\,1\,1\,0\,0\,1 \end{pmatrix}. \tag{3}$$

It is known that this diffusion layer has branch number 5 (see [12,1]). By referring to Table 1, it is an almost-optimal $\{0,1\}$-matrix.

4 Differential Bounds and Security Against Boomerang Attacks for Ciphers Based on $\{0,1\}$-Matrices

We shall now study SDS structures where the S-boxes are affine transforms of the inversion function over $GF(2^m)$ defined by: $S(x) = x^{-1}$ when $x \neq 0$ and $S(0) = 0$.

In order to apply Proposition 2, we need to find the difference distribution $DP^S(u \rightarrow j)$ where $u \in GF(2^m)$ is fixed and j varies over all of $GF(2^m)$, i.e. the difference distribution for each row of the difference table. Likewise, we need to find the difference distribution of the columns. It is well-known that the difference distribution table of $S^{-1}(x)$ is the transpose of the original difference distribution table. Since $S^{-1}(x) = S(x)$ for the inversion function, the difference distribution of the columns will be the same as that for the rows.

From the proof of [17, Proposition 6], we see that $S(x) + S(x+u) = j$, where u is fixed and j varies over $GF(2^m)$, can be reduced to a quadratic equation and it has 0 or 2 solutions for x in general. Only when m is even, we have 4 solutions for one value of j. Thus the inversion mapping has maximal differential 4 when m is even and 2 when m is odd. It is also well-known that the sum of each row of the difference distribution table should be 2^m. From this, we can easily get the difference distribution for each row of the inversion mapping shown in Table 2 in Appendix B.

Theorem 2. *Consider a SDS structure where the S-boxes are affine transforms of the inversion function over $GF(2^m)$ defined by: $S(x) = x^{-1}$ when $x \neq 0$ and $S(0) = 0$, and the diffusion mapping has branch number $Br(D) = k$. Then the differential probability of the SDS structure is:*

(i) $2^{(1-m)(k-1)} - 2^{(1-m)k+1} + 2^{(2-m)k}$ *when m is even;*
(ii) $2^{(1-m)(k-1)}$ *when m is odd.*

The proof of Theorem 2 can be found in Appendix A.4.

Remark 1. Note that when n is even, the upper bound for the differential probability from Proposition 3 is $(4/2^m)^{k-1} = 2^{(2-m)(k-1)}$. By simplifying the inequality:

$$2^{(2-m)k} - 2^{(1-m)k+1} < 2^{m-2}(2^{(2-m)k} - 2^{(1-m)k+1}),$$

we get the inequality:

$$2^{(1-m)(k-1)} - 2^{(1-m)k+1} + 2^{(2-m)k} < 2^{(2-m)(k-1)}.$$

This shows that the bound in Theorem 2 is better (lower) than that obtained from Proposition 3 when n is even. On the other hand, the upper bound of Theorem 2 and Proposition 3 is the same when n is odd.

4.1 Application on the E2 Cipher

The E2 cipher is a 12-round Feistel block cipher with a block size of 128 bits and a key size of 128, 192, or 256 bits [12]. There is also an initial transform (IT) that XORs a subkey, multiplies by a subkey and performs a byte permutation BP; and a final transformation (FT) that performs an inverse byte permutation BP^{-1}, divides by a subkey and XORs a subkey.

The nonlinear F-function in each Feistel round maps 64-bit to 64-bit with the help of two 64-bit subkeys $K^{(1)}$ and $K^{(2)}$. It is defined by:

$$F(x, K^{(1)}, K^{(2)}) = L(S(D(S(x \oplus K^{(1)})) \oplus K^{(2)})),$$

where $S : GF(2^8)^8 \to GF(2^8)^8$ is defined as:

$$S(x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7) = (s(x_0), s(x_1), s(x_2), s(x_3), s(x_4), s(x_5), s(x_6), s(x_7)).$$

Each $s : GF(2^8) \to GF(2^8)$ is affinely equivalent to the inversion function on $GF(2^8)$ and D is a 8-byte linear transform defined by the matrix in equation (3), which is known to have branch number 5 [1,12]. The final linear transform L is a 1-byte cyclic rotation over 8 bytes.

Before we go on to the analysis, we shall need the following standard result on the differential probability of Feistel ciphers.

Proposition 5. *[22] Consider a* 3*-round Feistel cipher with* $2w$*-bit block size and nonlinear function* $F : GF(2)^w \to GF(2)^w$*. If the maximal differential probability of* $F(x)$ *is* p*, then the maximal differential probability of the* 3*-round cipher is* p^2*.*

In [12], the authors derived the differential characteristic probability of E2 and concluded that it has practical security against differential cryptanalysis. Here we can give upper bounds for its differential probability in Theorem 3. The proof of the result can be found in Appendix A.5.

Theorem 3. *The differential probability of* 3 *rounds of the E2 cipher is at most* $2^{-55.39}$*.*

Thus to defend E2 against a stronger form of differential attack using true differentials, we can recommend a change of key after every 2^{55} encryptions.

Security of E2 against Boomerang Attack. There is also a stronger form of differential attack called boomerang attack [21]. It splits $R-1$ rounds ($R-2$ for Feistel ciphers) of an R-round block cipher into 2 shorter ciphers such that the differential probability of each part is known to be large, say with probability p for the differential $\alpha \rightarrow \beta$ for the first part and probability q for the differential $\gamma \rightarrow \delta$ for the second part. The distinguisher is the following boomerang process:

(i) Ask for the encryption of a pair of plaintexts (P_1, P_2) such that $P_1 \oplus P_2 = \alpha$ and denote the corresponding ciphertexts by (C_1, C_2).
(ii) Calculate $C_3 = C_1 \oplus \delta$ and $C_4 = C_2 \oplus \delta$, and ask for the decryption of the pair (C_3, C_4). Denote the corresponding plaintexts by (P_3, P_4).
(iii) Check whether $(P_3, P_4) = \alpha$.

For a random permutation, the probability that the last condition is satisfied is $2^{-blocksize}$. The probability that a quartet of plaintexts and ciphertexts satisfies the boomerang conditions is $(pq)^2$. Therefore, we have a distinguisher which distinguishes between the cipher being attacked and a random cipher if $(pq)^2 < 2^{-blocksize}$.

For the E2 cipher, 8 rounds of the cipher already has maximal differential characteristic probability $((2^{-6})^5)^2 \times ((2^{-6})^5)^2 \times (2^{-6})^5 = 2^{-150}$ which is less than 2^{-128}. Here we use the easily proven fact that there are at least 2 active F-functions for every 3 rounds and at least 1 active F-function every 2 rounds. Thus it is unlikely that an adversary can find a good differential over 8 rounds and any good differential is likely to involve 7 or less rounds. Thus when the adversary splits $12 - 2 = 10$ rounds into two sub-ciphers, they will each contain at least 3 rounds. By Theorem 3, we see that the differential probabilities p, q of the 2 sub-ciphers are at most $2^{55.39}$. Thus $(pq)^2 \leq 2^{-221.570} < 2^{-128}$ and E2 is secure against boomerang attack.

Remark 2. We have used the assumption that if the differential characteristic probability of R' rounds of a cipher is less than $2^{-blocksize}$, then it is not likely that a good differential over R' rounds can be found. This is in line with the common approach of practical provable security against differential cryptanalysis employed in the proofs of security of many ciphers like Camellia [1], AES [8], SQUARE [10], E2 [12], MCrypton [15] and SHARK [19]. Thus if our assumption is not true, then the approach is wrong because although we can prove that the differential characteristic probability is less than $2^{-blocksize}$, we can still find a differential with high probability to launch differential cryptanalysis. We shall also make the same assumption in the analysis of MCrypton against boomerang attack in Section 4.2.

Security of E2 Against Variants of Boomerang Attack. Since the boomerang attack requires adaptively chosen plaintexts and ciphertexts, many of the techniques that were developed for using distinguishers in key recovery attacks cannot be applied. As an alternative, the amplified boomerang attack [13] encrypts many plaintext pairs with the same input difference α and looks for right quartets which satisfy the requirements of the boomerang process. Out of x plaintext pairs, the number of right quartets is expected to be

$x^2 \cdot 2^{-blocksize+1} p^2 q^2$ [4]. For a random permutation, the expected number of right quartets is $x^2 \cdot 2^{-2 \cdot blocksize}$. Therefore, if $(pq)^2 > 2^{-blocksize+1}$, then we would count more quartets than random noise. For protection against the amplified boomerang attack, we would want to show that $(pq)^2 < 2^{-blocksize+1}$. Following the above argument, we know that $(pq)^2 \leq 2^{-221.570} < 2^{-128+1} = 2^{-127}$. Thus, E2 is also secure against the amplified boomerang attack.

Another variant of the boomerang attack is: suppose the initial and terminal differences, α and δ, are fixed while the intermediate differences, β and γ, are allowed to vary over index sets $\Lambda, \Omega \subseteq GF(2)^{blocksize}$ respectively, i.e. the attacker tries to find several differential paths with the same initial and terminal differences of high probability. Then he needs $\sum_{\beta \in \Lambda} \Pr^2(\alpha \to \beta) \sum_{\gamma \in \Omega} \Pr^2(\gamma \to \delta) > 2^{-blocksize}$ to get a good distinguisher for the attack to succeed. For E2, we know that for any $\alpha, \beta, \gamma, \delta$, $\Pr^2(\alpha \to \beta) \cdot \Pr^2(\gamma \to \delta) \leq 2^{-221.570}$. Thus the attacker must obtain at least $\lceil 2^{221.570}/2^{-128} \rceil + 1 = \lceil 2^{93.570} \rceil + 1 > 10^{28}$ high probability differential paths for the attack to work. It is highly improbable that the attacker will be able to find such a large number of useful differential paths and hence, this attack is unlikely to succeed.

4.2 Application on the MCrypton Cipher

The MCrypton cipher is a 12-round block cipher with a block size of 64 bits and a key size of 64, 96, or 128 bits [15]. Its structure is similar to that of the AES cipher [8] but it uses lightweight components suited for RFID applications. The plaintext is first written as a 4 by 4 array of nibbles and XORed to a 64-bit subkey. It then goes through 12 rounds of transformation where every round is composed of four operations:

$$\rho_K = \sigma_K \circ \tau \circ \pi \circ \lambda,$$

The substitution operation λ transforms each nibble by an affine transform of the inversion map on $GF(2^4)$. The linear map π multiplies each column of the array by one of the 4×4 $\{0,1\}$-matrices from Proposition 4 part (ii). The linear map τ transposes the 4 by 4 array. Finally, a 64-bit subkey is XORed with the array. The proof of the following Theorem 4 is given in Appendix A.6.

Theorem 4. *The differential probability of* 4 *rounds of the MCrypton cipher is at most* $2^{-22.62}$.

Thus, to defend MCrypton against a stronger form of differential attack using true differentials, we can recommend a change of key after every 2^{22} encryptions.

Security of MCrypton Against Boomerang Attack. For MCrypton, 8 rounds of the cipher should have differential characteristic probability at most $(2^{-2})^{32} = 2^{-64}$ if we follow the approach of AES. However, due to the careful choice of S-boxes and diffusion mappings of MCrypton in [15, Section 3], the authors proved that the differential characteristic probability is at most $2^{-96} < 2^{-64}$. Therefore it is unlikely that an adversary can find a good differential over 8 rounds and any good differential is likely to involve 7 or less rounds. Thus

when the adversary splits $12-1=11$ rounds into two sub-ciphers, they will each contain at least 4 rounds. By Theorem 4, we see that the differential probabilities p, q of the 2 sub-ciphers are at most $2^{-22.62}$. Thus $(pq)^2 \leq 2^{-90.49} < 2^{-64}$ and MCrypton is secure against boomerang attack.

Security of MCrypton Against Variants of Boomerang Attack. Since $(pq)^2 \leq 2^{-90.59} < 2^{-64+1} = 2^{-63}$, MCrypton is also secure against the amplified boomerang attack. For the second variant of the boomerang attack where only the initial and terminal differences are fixed while the intermediate differences are allowed to vary, at least $\lceil 2^{90.59}/2^{64} \rceil + 1 = \lceil 2^{26.59} \rceil + 1 > 10^8$ high probability differential paths are required. Again, it is unlikely that the attacker will be able to find that many useful differential paths.

5 On Differential Probability, Universal Hash Functions and Message Authentication Codes

Let $H : [GF(2)^w]^* \rightarrow GF(2)^w$ be a family of functions. The probabilities below, denoted by $\Pr_{h \in H}[\cdot]$, are taken over the choice of $h \in H$.

Definition 7. *H is a Δ-*universal family of hash functions *if for all $x, y \in [GF(2)^w]^*$ with $x \neq y$ and all $a \in GF(2)^w$, $\Pr_{h \in H}[h(x) - h(y) = a] = 2^{-w}$.*

Definition 8. *H is an ϵ-*almost-Δ-universal *(ϵ-AΔU) family of hash functions if $\Pr_{h \in H}[h(x) - h(y) = a] \leq \epsilon$.*

It is well-known that $\epsilon \geq 2^{-w}$ (see [20]). Universal hash functions can be used to construct *Message Authentication Codes* (MAC) via the Wegman-Carter approach [23]. The MAC tag is given by the value $h(msg)$ exclusive-or-ed with the one-time-pad OTP as follows:

$$MAC_{h,OTP}(msg) = h(msg) \oplus OTP$$

where h is a randomly chosen hash function from the family H and OTP is a random one-time-pad. The communicating parties must share the secret key pair (h, OTP) in this scenario. However, it is not practical to generate one-time-pads long enough to handle long messages. In [6], Brassard proposed that we substitute the one-time-pad encryption with a computationally secure encryption scheme, for example, AES.

In [14], the authors constructed the following universal hash functions based on functions with low maximal differential.

Proposition 6. *([14], Theorem 1]) Let $f : GF(2)^w \rightarrow GF(2)^w$ have maximal differential Δ_f. Let $x = (x_1, \ldots, x_r)$ and $msg = (msg_1, \ldots, msg_r)$ where $x_i, msg_i \in GF(2)^w$. The* function sum hash (FSH) family of functions *defined by $FSH = \{h_x : [GF(2)^w]^r \rightarrow GF(2)^w | x \in [GF(2)^w]^r\}$ where $h_x(msg) = \sum_{i=1}^{r} f(msg_i + x_i)$ is an ϵ-AΔ universal family of hash functions with $\epsilon \leq \frac{\Delta_f}{2^w}$.*

By applying Proposition 6 to Proposition 3 and Theorem 2, we get the following result.

Proposition 7. *Let* $h_x : GF(2)^{mnr} \rightarrow GF(2)^{mn}$ *be a FSH based on a SDS structure defined by*

$$h_x(msg) = \sum_{i=1}^{r}(S(D(S(msg_i + x_i))),$$

where $x = (x_1, \ldots, x_r)$, $msg = (msg_1, \ldots, msg_r)$, $D : GF(2^m)^n \rightarrow GF(2^m)^n$ *has branch number* k *and* $S(\cdot)$ *is a layer of* n m*-bit S-boxes. Then* $h_x(msg)$ *is an* ϵ*-*$A\Delta$ *universal family of hash functions with:*

(i) $\epsilon \leq p^{k-1}$ *if the maximal differential probability of the S-boxes is* p*;*
(ii) $\epsilon \leq 2^{(1-m)(k-1)} - 2^{(1-m)k+1} + 2^{(2-m)k}$ *if the inversion S-box on* $GF(2^m)$ *is used where* m *is even;*
(iii) $\epsilon \leq 2^{(1-m)(k-1)}$ *if the inversion S-box on* $GF(2^m)$ *is used where* m *is odd.*

Proposition 7 can be viewed as an improvement over [14, Theorem 6] where the authors only approximated the upper bound of the universal hash function by the differential characteristic probability p^k, which is too high. By using the above two-round structure, we may be able to use it to compute a message authentication code (MAC) based on:

$$MAC_{K,x}(msg) = Enc_K(\sum_{i=1}^{r}(S(D(S(msg_i + x_i))))),$$

where $Enc(\cdot)$ is a R-round block cipher where each round XORs a subkey, applies the S-box layer S and then, the diffusion layer D. In this way, we get a MAC which is $R/2$ times faster than encryption and is easily parallelizable. Based on the upper bound on ϵ, we can restrict the number of MAC computations to less than $\sqrt{\epsilon^{-1}}$ when a change of MAC key is needed, so as to protect against forgery attacks. In this case, the MAC key consists of the encryption key K and a sequence of secret values $x_i \in GF(2)^{mn}$ which are to be shared between the sender and receiver. It may not be feasible to generate and share long strings of secret values $x_1, x_2, x_3, \ldots$ In [14], it is suggested that a single secret value $x_1 \in GF(2)^{mn}$ be chosen to seed a LFSR of length mn bits to produce $x_2, x_3, \ldots$

5.1 Applications to Ciphers Used in Practice

By applying Proposition 6 where f is taken to be the 3-round E2 cipher, we can construct an E2-based MAC as follows:

$$MAC_{K,x}(msg) = \text{E2}_K(\sum_{i=1}^{r} \text{3-Round-E2}(msg_i + x_i)),$$

where $x = (x_1, \ldots, x_r)$ and $msg = (msg_1, \ldots, msg_r)$. From Theorem 3, we see that the above MAC is based on a universal hash function with collision probability at most $\epsilon \leq 2^{-55.39}$. Thus we require a change of MAC key before $\sqrt{\epsilon^{-1}} \approx 2^{27.7}$ MAC computations. Moreover, the above MAC is $12/3 = 4$ times faster than CBC-MAC based on 12-round E2.

Similarly, by applying Proposition 6 where f is taken to be the 4-round MCrypton cipher, we can also construct a MCrypton-based MAC as follows:

$$MAC_{K,x}(msg) = \text{MCrypton}_K(\sum_{i=1}^{r} \text{4-Round-MCrypton}(msg_i + x_i)),$$

From Theorem 4, we see that the above MAC is based on a universal hash function with collision probability at most $\epsilon \leq 2^{-22.62}$. Thus we require a change of MAC key before $\sqrt{\epsilon^{-1}} \approx 2^{11.3}$ MAC computations. Moreover, the above MAC is $12/4 = 3$ times faster than CBC-MAC based on 12-round MCrypton.

Remark 3. Note that our universal-hash based MAC is parallelizable while CBC-MAC is not. So if we can have N copies of the E2 cipher, for example, then our MAC will be $4N$ times faster than E2-based CBC-MAC.

Remark 4. Note that we could have included the subkeys in the reduced-round ciphers used to define the universal hash function. This will not affect the probability bound of the universal hash function and heuristically, do give a more secure MAC as the secret key K is not just used in the final encryption, but also in the compression of every message block msg_i.

6 Conclusion

In this paper, we proved an upper bound for the branch numbers of $\{0,1\}$-matrices. Furthermore, we showed that they are never MDS and are almost-MDS only when $n = 2, 3$, or 4. The ciphers E2, Camellia, and MCrypton were found to employ $\{0,1\}$-matrices which are optimal or almost-optimal. We also used Park's result on the differential probability of the SDS structure in [18] to obtain a general formula for the differential probability of such a structure which uses an affine transform of the inverse S-box in its S-box layer. With this formula, we were able to prove the differential probability of E2 and MCrypton, as well as their resistance against boomerang attack and its variants. Our results provide new direction in the analysis of block ciphers based on $\{0,1\}$-matrices or non-MDS matrices and affine transforms of inverse S-boxes against differential-based attacks. Finally, we improved on the upper bound for a FSH based on a SDS structure. Two MACs based on E2 and MCrypton, faster than CBC-MAC and with provable collision probability, were also proposed.

References

1. Aoki, K., Ichikawa, T., Kanda, M., Matsui, M., Moriai, S., Nakajima, J., Tokita, T.: Camellia: A 128-bit Block Cipher Suitable for Multiple Platforms - Design and Analysis. In: Stinson, D.R., Tavares, S. (eds.) SAC 2000. LNCS, vol. 2012. Springer, Heidelberg (2001)
2. Barreto, P.S.L.M., Rijmen, V.: The WHIRLPOOL Hashing Function. Primitive submitted to NESSIE, revised on May 2003 (September 2000), http://paginas.terra.com.br/informatica/paulobarreto/WhirlpoolPage.html
3. Biham, E., Dunkelman, O., Keller, N.: The Rectangle Attack - Rectangling the Serpent. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 340–357. Springer, Heidelberg (2001)

4. Biham, E., Dunkelman, O., Keller, N.: Related-Key Boomerang and Rectangle Attack. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 507–525. Springer, Heidelberg (2005)
5. Biham, E., Shamir, A.: Differential Cryptanalysis of DES-like Cryptosystems. Journal of Cryptology 4 (1991)
6. Brassard, G.: On computationally secure authentication tags requiring short secret shared keys. In: Crypto 1983, pp. 79–86. Springer, Heidelberg (1983)
7. Daemen, J., Rijmen, V.: The Wide Trail Strategy. In: Honary, B. (ed.) Cryptography and Coding 2001. LNCS, vol. 2260, pp. 222–238. Springer, Heidelberg (2001)
8. Daemen, J., Rijmen, V.: The Design of Rijndael: AES, The Advanced Encryption Standard. Springer, Heidelberg (2002)
9. Daemen, J., Govaerts, R., Vandewalle, J.: Correlation Matrices. In: Preneel, B. (ed.) FSE 1994. LNCS, vol. 1008, pp. 275–285. Springer, Heidelberg (1995)
10. Daemen, J., Knudsen, L., Rijmen, V.: The Block Cipher Square. In: Biham, E. (ed.) FSE 1997. LNCS, vol. 1267, pp. 149–165. Springer, Heidelberg (1997)
11. Hong, S., Lee, S., Lim, J., Sung, J., Cheong, D., Cho, I.: Provable Security against Differential and Linear Cryptanalysis for the SPN Structure. In: Schneier, B. (ed.) FSE 2000. LNCS, vol. 1978, pp. 273–283. Springer, Heidelberg (2001)
12. Kanda, M., Moriai, S., Aoki, K., Ueda, H., Takashima, Y., Ohta, K., Matsumoto, T.: E2 - A New 128-bit Block Cipher. IEICE Transactions Fundamentals - Special Section on Cryptography and Information Security, vol. E83-A no. 1, pp. 48-59 (2000)
13. Kelsey, J., Kohno, T., Schneier, B.: Amplified Boomerang Attacks Against Reduced-Round MARS and Serpent. In: Schneier, B. (ed.) FSE 2000. LNCS, vol. 1978, pp. 75–93. Springer, Heidelberg (2001)
14. Khoo, K., Heng, S.H.: New Constructions of Universal Hash Functions based on Function Sum. In: Gavrilova, M.L., Gervasi, O., Kumar, V., Tan, C.J.K., Taniar, D., Laganá, A., Mun, Y., Choo, H. (eds.) ICCSA 2006. LNCS, vol. 3982, pp. 416–425. Springer, Heidelberg (2006)
15. Lim, C.H., Korkishko, T.: mCrypton - A Lightweight Block Cipher for Security of Low-Cost RFID Tags and Sensors. In: Song, J.-S., Kwon, T., Yung, M. (eds.) WISA 2005. LNCS, vol. 3786, pp. 243–258. Springer, Heidelberg (2006)
16. Matsui, M.: Linear Cryptanalysis Method for DES Cipher. In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 386–397. Springer, Heidelberg (1994)
17. Nyberg, K.: Differentially Uniform Mappings for Cryptography. In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 55–64. Springer, Heidelberg (1994)
18. Park, S., Sang, S.H., Lee, S., Lim, J.: Improving the Upper Bound on the Maximum Differential and the Maximum Linear Hull Probability for SPN Structures and AES. In: Johansson, T. (ed.) FSE 2003. LNCS, vol. 2887, pp. 247–260. Springer, Heidelberg (2003)
19. Rijmen, V., Daemen, J., Preneel, B., Bosselars, A., Win, E.D.: The Cipher Shark. In: Gollmann, D. (ed.) FSE 1996. LNCS, vol. 1039, pp. 99–111. Springer, Heidelberg (1996)
20. Stinson, D.R.: On the connections between universal hashing, combinatorial designs and error-correcting codes. In: Congressus Numerantium, vol. 114, pp. 7–27 (1996)
21. Wagner, D.: The Boomerang Attack. In: Knudsen, L.R. (ed.) FSE 1999. LNCS, vol. 1636, pp. 156–170. Springer, Heidelberg (1999)
22. Wallen, J.: Design Principles of the KASUMI Block Cipher, http://citeseer.ist.psu.edu/wallen00design.html
23. Wegman, M.N., Carter, J.L.: New hash functions and their use in authentication and set equality. Journal of Computer and System Sciences 22(3), 265–279 (1981)

A Proofs

A.1 Proof of Theorem 1

Proof. Suppose A has branch number k, where $k > (2n+4)/3$. Let v be a vector of weight 1. Then

$$wt(Av) \geq k - 1 > (2n + 1)/3.$$

On the other hand, if v is a vector of weight 2, then

$$wt(Av) \geq k - 2 > (2n - 2)/3.$$

Thus the XOR-sum of any two columns of A has weight greater than $(2n-2)/3$.

These two facts contradict each other: if v_1 and v_2 have weight greater $(2n + 1)/3$, then each of v_1 and v_2 has less than $(n-1)/3$ zeroes. For a bit of $v_1 \oplus v_2$ to be 1, exactly one of the corresponding bits of v_1 or v_2 must be 0. Thus $v_1 \oplus v_2$ has less than $2(n-1)/3 = (2n-2)/3$ ones, which is a contradiction. □

A.2 Proof of Corollary 1

Proof. For $n \geq 2$, it is easy to see that the upper bound from Theorem 1: $\frac{2n+4}{3}$ is always less than $n+1$, the MDS bound. It is also easy to show that the upper bound $\frac{2n+4}{3}$ is at least as large as n, the almost-MDS bound, when $n \leq 4$. □

A.3 Proof of Proposition 4

Proof. To prove part (i). It is easy to see that A has branch number 2 when $n = 2$.

For the case $n = 3$: If the input has weight 1, then the output which corresponds to a column of the matrix will have weight 2. If the input has weight 2, then the output which corresponds to a linear combination of two columns will have weight at least 2. If the input has weight 3, then there will definitely be ≥ 4 non-zero entries in the total (input and output) as the output must be non-zero. Thus the branch number of A is 3.

For the case $n \geq 4$: If the input has weight 1, then the output which corresponds to a column of the matrix will have weight $n - 1 \geq 3$. If the input has weight 2, then the output which corresponds to a linear combination of two columns will have weight at least 2. If the input has weight 3, then the output which corresponds to a linear combination of three columns will have weight at least 1. If the input has weight ≥ 4, then there will be ≥ 5 non-zero entries in total since the output is non-zero. Thus the branch number of A is 4.

It is easy to see that permuting the rows and columns of a matrix will preserve its branch number, thus part (ii) follows naturally from part (i).

Part (iii) is verified by a computer search over all 4×4 $\{0, 1\}$-matrices. □

A.4 Proof of Theorem 2

Proof. Because the S-boxes used are affine transforms of the inversion function, they have the same difference distribution. Moreover, every row and every column of the inversion function have the same distribution as explained above; thus we just need to consider one row in the difference table. This allows us to simplify the upper bound in Proposition 2 to $\sum_{j=1}^{2^m-1} DP^{S_i}(u \to j)^k$, $u \neq 0$.

(i) When n is even:

$$\sum_{j=1}^{2^m-1} DP^{S_i}(u \to j)^k \leq (2/2^m)^k(2^{m-1}-2) + (4/2^m)^k$$
$$= 2^{(1-m)(k-1)} - 2^{(1-m)k+1} + 2^{(2-m)k}.$$

(ii) When n is odd:

$$\sum_{j=1}^{2^m-1} DP^{S_i}(u \to j)^k \leq (2/2^m)^k \times 2^{m-1}$$
$$= 2^{(1-m)(k-1)}.$$ □

A.5 Proof of Theorem 3

Proof. Because the inversion map is used and the diffusion map has branch number 5, we can apply Theorem 2 with $m = 8$ and $k = 5$ to get an upper bound for the differential probability of the SDS structure in the F-function:

$$DP_{SDS} \leq 2^{(1-8)(5-1)} - 2^{(1-8)5+1} + 2^{(2-8)5} \approx 2^{-27.696}.$$

The final linear transform L does not influence the maximal differential probability of the F function since it simply rotates the bytes. Therefore, the maximal differential probability of F is $2^{-27.696}$. By applying Proposition 5, the maximal differential probability of 3 rounds of E2 is at most $(2^{-27.696})^2 = 2^{-55.392}$. □

A.6 Proof of Theorem 4

Proof. Here we base much of the proof that follows on the wide trail strategy (see [7, Theorem 3], [8, Theorem 9.5.1]). The design of MCrypton follows the design principle in AES [8] where τ is a diffusion optimal mapping. After an admissible re-arrangement of the operations in MCrypton, we can view the cipher as alternating between $\pi \circ \lambda$ and $\tau \circ \pi \circ \tau \circ \lambda$. The linear map $\tau \circ \pi \circ \tau$ has the same branch number as π but it acts on bundles of size 4-nibble (16-bit). Since the branch number of π is 4 and τ is a diffusion optimal transposition of bundles, $\tau \circ \pi \circ \tau$ also has branch number 4.

Each bundle over a $\lambda \circ \pi \circ \lambda$ transformation is a 16-bit SDS structure consisting of the linear map π sandwiched between two layers of four 4-bit S-boxes. Since

π has branch number 4 by Proposition 4 part (ii) and each S-box is an affine transform of the inversion function on $GF(2^4)$, we can apply Theorem 2 with $m = 4$, $k = 4$. The maximal differential probability is:

$$DP_{SDS} \leq 2^{(1-4)(4-1)} - 2^{(1-4)4+1} + 2^{(2-4)4} \approx 2^{-7.54}.$$

Next, we can view 4 rounds of MCrypton as $\tau \circ \pi \circ \tau$ sandwiched between two layers of four bundles where each bundle has differential probability at most $2^{-7.54}$. By Proposition 3, the differential probability of 4 rounds is upper bounded by $(2^{-7.54})^{4-1} = 2^{-22.62}$. □

B Tables

Table 1. Upper Bound for the Branch Number in Theorem 1

Size of n for $n \times n$ $\{0,1\}$-Matrix	2	3	4	5	6	7	8	9	10
Upper Bound of Branch Number	2	3	4	4	5	6	6	7	8

Table 2. Difference Distribution for Each Row of the Difference Distribution Table for the Inversion Mapping on $GF(2^m)$

Difference	Frequency (m even)	Frequency (m odd)
0	$2^{m-1} + 1$	2^{m-1}
2	$2^{m-1} - 2$	2^{m-1}
4	1	0

C Diagrams

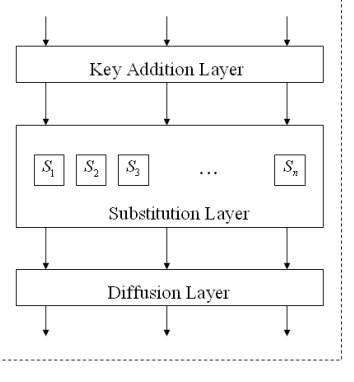


Fig. 1. One round of a SPN structure

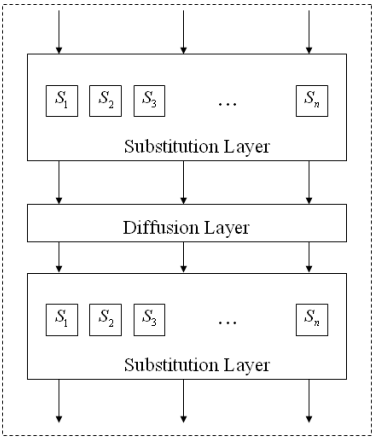


Fig. 2. SDS function

HAPADEP: Human-Assisted Pure Audio Device Pairing

Claudio Soriente, Gene Tsudik, and Ersin Uzun

Computer Science Department,
University of California, Irvine
{csorient,gts,euzun}@ics.uci.edu

Abstract. The number and diversity of personal electronic gadgets have been steadily increasing but there has been fairly little progress in secure pairing of such devices. The pairing challenge revolves around establishing on-the-fly secure communication without any trusted (on- or off-line) third parties between devices that have no prior association. One basic approach to counter Man-in-the-Middle (MiTM) attacks in such setting is to involve the user in the pairing process. Previous research yielded some interesting secure pairing techniques, some of which ask too much of the human user, while others assume availability of specialized equipment (e.g., wires, photo or video cameras) on personal devices. Furthermore, all prior methods assumed an established insecure channel over a common digital (human-imperceptible) communication medium, such as infrared, 802.11 or Bluetooth.

In this paper we introduce a very simple technique called HAPADEP (Human-Assisted Pure Audio Device Pairing). HAPADEP uses the audio channel to exchange both data and verification information among devices without requiring any other means of common electronic communication. Despite its simplicity, a number of interesting issues arise in the design of HAPADEP. We discuss design and implementation highlights as well as usability features and limitations.

Keywords: User-Aided Security, Secure Device Pairing, Authentication Protocols, Secure First Connect, Man-in-the-Middle attacks.

1 Introduction and Motivation

The popularity of sophisticated personal devices, such as PDAs, multimedia players, cameras and smartphones, has prompted the need for security mechanisms specifically tailored for such devices. One of the main challenges is the problem referred to as: *Secure First Connect*, *Secure Initialization* or *Secure Device Pairing*.[1] Regardless of the name, this problem entails the establishment of a secure communication channel between previously unassociated devices.

Traditional cryptographic means of establishing secure communication channels (e.g., authenticated key exchange protocols) are generally unsuitable for

[1] In this paper, we use the term "device pairing" to refer to the problem at hand.

T.-C. Wu et al. (Eds.): ISC 2008, LNCS 5222, pp. 385–400, 2008.

secure device pairing. This is because mutually unfamiliar devices have no prior context and no common point of trust: no on-line Trusted Third Party (TTP), no off-line Certification Authority (CA), no Public Key Infrastructure (PKI) and, of course, no common secrets.

Since no pre-shared secret or other means of authentication exists between two unfamiliar devices and the communication transpires over some human-imperceptible medium (e.g., infrared or radio), user assistance in the secure device pairing process is simply unavoidable. This is because of the very real threat of so-called Man-in-the-Middle (MiTM) attacks. A MiTM attack can occur whenever unauthenticated communication is involved. One of the best-known examples is the textbook Diffie-Hellman Key Exchange protocol [29]. The security research community has recognized, and begun responding to, this challenge starting in the late 1990-s, i.e., the pioneering work of Stajano, et al. [8]. A number of techniques have been proposed since then, varying greatly in the assumptions about device features, degree and nature of user involvement as well as environmental factors.

Several standardization bodies also recognized the importance of the problem and have begun working on specifying more usable and more secure procedures for device pairing. Wi-Fi Alliance has released specifications for Wi-Fi Protected Setup [4]. Microsoft has released specifications for Windows Connect Now-NET [22], Bluetooth Special Interest Group has released specifications on Simple Pairing [11] and the Universal Serial Bus (USB) forum has released the specifications for Wireless USB Association Models [30].

Despite significant recent progress, the design space of the device pairing problem has not been fully explored. In particular, two issues remain unaddressed:

- Denial-of-Service (DoS): unlike MiTM attacks, DoS attacks aim to prevent communication. In the device pairing setting, the goal of a DoS attack is to preclude the two devices from establishing a secure channel. Such an attack is trivial to mount: the adversary only needs to jam the interface of one or both devices. This is easy with Bluetooth or 802.11, but a little harder (since it requires line-of-sight) – yet still doable – with infrared. The gist of the problem is two-fold:
  (1) the inability of the human user to detect a DoS attack while it is taking place, and,
  (2) even if a DoS attack is detected, the difficulty of determining its source
- Common Channel: all prior techniques require the existence (and set-up) of a common means of electronic communication between two devices. The devices must have at least one interface in common, whether wired (as in [8]) or wireless (as in [13,10,23,14,15]). This can be problematic, for two reasons:
  (1) The two devices might not (at least at the time of pairing) have a common interface, e.g., one only has a Bluetooth interface, while the other – 802.11. Why pair such devices? One reason could be because they would later connect to the Internet via respective interfaces and need to communicate securely.
  (2) Wireless interfaces typically take some time and effort to configure. This can be a real challenge for an ordinary user. For example, in case of

802.11-equipped devices, each must be put into the ad hoc mode and an ad hoc network must be manually configured. Similarly, infrared (IrDA) interfaces must be manually activated on both devices; IrDA also requires line-of-sight alignment. In case of Bluetooth, one device needs to be *discoverable* and then the other must *discover* it. This is tricky if there are many Bluetooth-enabled devices around, and may turn into a real headache if multiple devices share the same default (or common) name. Appendix A shows some screen-shots from the confusing procedures involved in configuring 802.11 and Bluetooth for initial ad-hoc communication.

The work described in this paper attempts to fill the gap left by prior techniques. The proposed protocol – HAPADEP or Human-Assisted Pure Audio Device Pairing – obviates the need for a common human-imperceptible communication channel and its set-up. It uses the audio channel for communicating both protocol messages and human-perceived authentication/verification information. HAPADEP takes advantage of the fact that many modern devices are equipped with audio in/out interfaces, i.e. a speaker and a microphone, and such interfaces are very inexpensive to add for the others. In our usability evaluations, HAPADEP has shown itself to be easy to use and preferred by the users over most other techniques. Also, as we describe further in the paper, HAPADEP offers some natural (albeit limited) means of protection against both DoS and MiTM attacks.

The rest of this paper is organized as follows: we survey related work in section 2 and describe the HAPADEP protocol in section 3. We describe two HAPADEP variants in section 4 and discuss their respective usability factors in section 5. Section 6 discusses the security and some other aspects of HAPADEP and section 7 finalizes the paper with summary and future work.

2 Related Work

There is a fairly large body of relevant prior work on the general topic of secure device pairing.

The earliest work by Stajano, et al. [8] made a seminal contribution by bringing the problem into the spotlight. The proposed techniques, however, required the use of standardized physical interfaces and cables. The follow-on work by Balfanz, et al. [5] and Feeney, et al. [7] made progress by suggesting the use of infrared communication as the human-verifiable side-channel. Though timely in its day, this approach is no longer viable since: (1) few modern devices are equipped with IrDA interfaces and (2) the infrared channel itself is not fully immune to DoS and MiTM attacks.

Another early approach involves image comparison. It encodes the authentication data into images and asks the user to compare them on two devices. Prominent examples include “Snowflake” [9], “Random Arts Visual Hash” [3] and “Colorful Flag” [6]. Since these methods require both devices to have displays with sufficiently high resolution, applicability is limited to high-end devices,

such as: laptops, PDAs and certain cell phones. A more practical approach, based on SAS protocols [24,17], suitable for simpler displays and LEDs has been investigated by Roth, et al. [26].

The *Seeing-is-Believing* technique by McCune, et al. [13] uses the visual channel to perform secure device pairing. One device sends its public key to the other through a human-imperceptible channel (such as 802.11) and, at the same time, displays a visual encoding of the public key in the form of a bar code. The receiver device, with the help of the user, takes a picture of the bar code and compares it with the one computed locally. The user is not required to recognize pictures, but it requires at least one device to have a photo camera, and the quality of the picture, either printed or displayed, to be quite good. For bidirectional authentication the above sequence must be repeated twice.

Saxena, et al. [23] considered a variation of Seeing-is-Believing [13] method by showing how to achieve secure pairing if one device is equipped with a video camera, while the other has at least a single LED. In this method, the device equipped with an LED uses its "blinking" capability to transmit the hash. The device with a camera records the blinking pattern, extracts the hash and compares with the hash computed as a result of the protocol. If they match, it asks the user to accept on the other device; otherwise it asks user to abort.

Another device pairing approach – *Loud-and-Clear* [10] – uses audio as its human-perceptible channel. Loud-and-Clear protocols involve two devices exchanging their keys and computing the hash of the exchanged cryptographic material. The hash is then translated in a syntactically correct English-like "Madlib" (non-sensical) sentence that can be played by one of the devices and showed on a text display on the other: the user compares the sequences and verifies the key exchange. The authors consider many other scenarios and variations of the protocol, but the main contribution is the ability to perform secure device pairing between a device equipped with a speaker and another equipped with a simple (text) display. Moreover, the generation of syntactically correct text helps the user in the verification process.

Yet another approach –"Button-Enabled Device Authentication (BEDA)" – by Soriente, et al. [27], suggests a pairing method for interface constrained devices. To accommodate wide variety devices, this method has several variants: "LED-Button", "Vibration-Button" and "Button-Button". In the first two variants, the sending device blinks its LED (or vibrates) and the user presses a button on the receiving device at each such event. String is encoded as the delay between consecutive blinks (or vibrations). In the Button-Button variant the user simultaneously presses buttons on both devices and random user-controlled inter-button-press delays are used as a means of establishing a common secret. Saxena, et al. [25] also proposed a pairing method suitable for constrained devices. The proposed method is based on synchronized audio-visual patterns and involve users comparing them, e.g., in the form of "beeping" and "blinking", transmitted as simultaneous streams. The advantage of these methods is that they require devices to only have very basic interface such as buttons, LEDs or a basic beeper.

There have been other proposals suggesting the use of technologies that are more expensive and less common. Kinberg, et al. suggested an approach requiring RF and ultrasound receiver/transmitters on both devices in their earlier work [15] and laser technology (requires each device to be equipped with a laser transreceiver) in their more recent proposal [14]. Holmquist, et al. [12] and Mayrhofer et al. [21] proposed the use of a common movement pattern as the security initiator when the two devices are shaken together. This requires both devices to be equipped with two-axis accelerometers; it is also unsuitable for physically large devices.

All aforementioned techniques adhere to one common design element: they use two channels to perform the pairing process. The primary human-imperceptible channel is used to exchange the cryptographic material, and, then, a secondary human-perceptible channel is used to verify the integrity of the process. The main drawback of this approach is the requirement that devices must have a common communication channel, such as 802.11a/b/g/n, Bluetooth, IrDA, and WiBro. In some scenarios, a common communication channel might not be available at the time of pairing and, even when available, configuring it to establish communication is time-consuming and cumbersome.

3 HAPADEP: General Operation

The HAPADEP protocol relies only on the audio channel. A speaker and a microphone are the only required device features. (In fact, in its simplest, unilateral flavor, HAPADEP eliminates the need for a microphone in one of the devices.) We consider both unilateral and bilateral key exchange protocol flavors. When talking about the former, we use the term *personal* to denote the device receiving the public key and *target* to denote the device sending (delivering) its public key. In the bilateral protocol, we use a more generic term *peer*.

A unilateral protocol is said to be *verifiable* if the user, at the end of the protocol execution, is sure that one device has correctly received the public key sent by the other device (The same property trivially extends to bilateral key exchange). The notion of verifiable key exchange is somewhat different from the notion of authenticated key exchange as the latter guarantees the *owner* of the received cryptographic material, while the former guarantees its *sender*. However, since pairing scenarios involve direct exchange of information – rather than delivery via intermediaries – verified key exchange is sufficient to ensure security.

The HAPADEP protocol consists of two phases, as shown in Figure 1. During the first *transfer* phase the target device sends its public key (and any other cryptographic material) to the personal device; during this phase, the user is only responsible for triggering the execution of the protocol, i.e., pushing a button on each device. To transfer the key over the audio channel, the target encodes its key using the codec and plays the resulting audio sequence. The personal device records the audio sequence and decodes it to retrieve the key. In a bilateral key exchange protocol, the transfer phase is repeated with the devices that

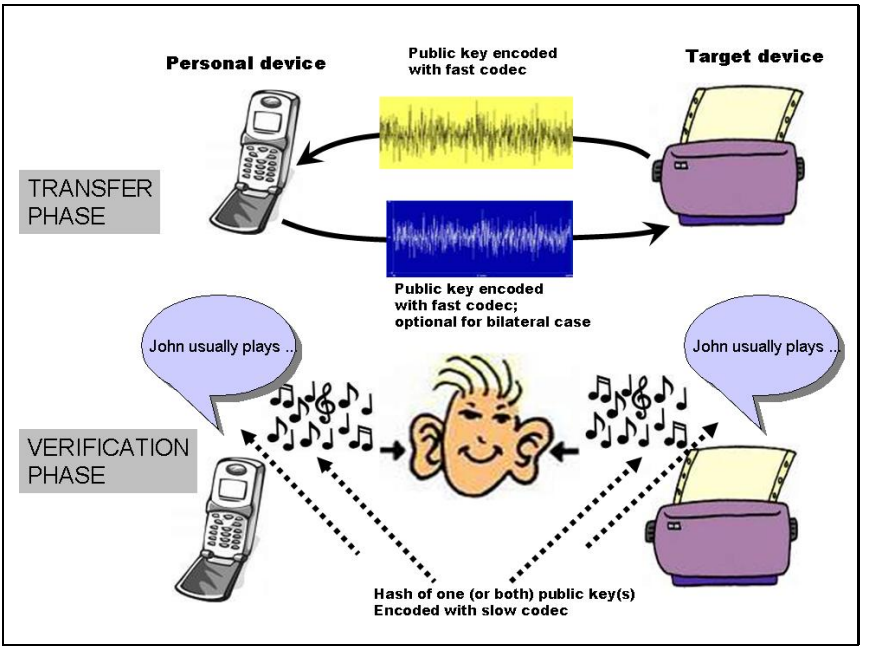


Fig. 1. HAPADEP Operation

automatically switch roles. Since encoded public keys are fairly large (several hundred bits), encoding is done using a *fast* codec that provides faster data transmission rate. Although the faster transmission rate has a side-effect of producing rather unpleasant-sounding audio, the transfer only takes about 3 seconds (6 seconds for the bilateral case). Moreover, the human user is not expected to pay close attention or be actively involved at this stage.

During the second *verification* phase the user is directly involved in verifying that the key exchange is secure and successful. A relatively *slow* codec is used to encode the digest of the cryptographic material exchanged in the *transfer* phase. The bit rate of this codec can be much lower than the *faster* codec used in the first phase[2], however, the output needs to be pleasant to the human ear and easily recognizable. In the verification phase, each device plays the audio sequence created using the *slow* codec and waits for the user to decide whether the two sequences match. The user is expected to listen to both sequences carefully and indicate (e.g., by pressing a key) a match or lack thereof.

It might seem like the verification phase is unnecessary. Indeed, we could imagine a simpler protocol which would avoid the verification phase relying instead on the human user to detect exactly *which* device is playing during the transfer phase: if the target is actually playing the audio sequence, any audible MiTM will be noticed. However, two issues would arise: First, the output of

[2] Since verification data is small, the low bit rate is not a concern here.

the fast codec is not pleasant to the human ear which might discourage users from paying attention and identifying the audio source. And even if they pay attention or the output is made pleasant, asking to decide if there were any other audio source producing a similar sound is problematic as there may be many such sources in a fairly crowded city spot. Second, it is important to verify the successful termination of the protocol and give a satisfying proof to the user. Note that the user can be sure that the target device is playing but s/he cannot tell whether or what the personal device is recording.

Another naïve approach might be to use the slow (more pleasant-sounding) codec for the cryptographic exchange and play the recorded audio on the receiver. Then, the user would be expected to determine whether the sequentially played sequences by the devices are the same. However, the amount of data encoded as audio would be much larger which would result in long audio sequences, making it tiresome for the user.

Since both devices are capable of playing, recording and comparing audio sequences, it might also seem possible to have the devices themselves (without any user involvement) check whether the two audio sequences played during the verification phase are identical. This approach would remove all burden from the user. However, it would be easy for a malicious device to participate to both transfer and verification phases, pretending to be the target device. In other words, the presence of the user telling which device (among several) is playing is an essential feature termed by Balfanz, et al. as *Demonstrative Identification* [5].

Since the protocol must be fast, secure and usable, we claim that the need for two phases and two different codecs is well-justified.

4 Implementation

In the implementation of HAPADEP, choosing the right codecs is crucial. In the case of *fast* codec, the two main requirements are reliability (low error rate) and high bit-rate. In this respect, any reliable and fast codec that can cope with reasonable amount of background noise can be used in the implementation. The *slow* codec, on the other hand, has to be chosen very carefully since it directly affects the usability and the security of the protocol. In HAPADEP, user's ability to tell whether the verification sequences are the same one or they are different is crucial for the security of the protocol. So the output has to be easily recognizable by a human user.

In our implementation, the *fast* codec is based on the results of the *Digital Voices* project at PARC [18,20,19]. 240 bits are encoded in a 3.4-second midi audio sequence where the first 160 bits represent the actual public key (in the EC-DSA cryptosystem) and the last 80 bits are for error checking. The *Bouncy Castle* [1] crypto package is used for cryptographic computations.

For the human-verifiable audio generation (*slow* codec), we implemented (and experimented with) two different approaches:

- Using pleasant-sounding short melodies
- Using grammatically correct MadLib sentences

The implementations details for each approach are as follows:

Generating a Melody: The codec uses the hash of the exchanged cryptographic protocol data to produce a MIDI score played by a piano. Using this codec, playing time for the resulting MIDI sequence (generated from a 80 bit input) takes about 5.6-seconds. For each byte of the input, the first four bits are used to select a specific *symbol* to add to the score from among the seven major chords, the seven minor chords, pause and sustain; the second four bits together with the previous ones and the present octave, are used to select the octave at which the note is played. The result is an easily recognizable audio sequence, very similar to sounds produced by a child playing a toy piano. Each device can replay the sequence multiple times so that the user (if desired) can make sure that the two devices play the same (or different) sequences.

Generating a Sentence: Another way to convert cryptographic data to human-verifiable audio is by producing grammatically correct sentences. The logic is similar to the MadLib game which was also used in the *Loud-and-Clear* [10] device pairing technique. We employed the same MadLib generation method to create grammatically correct but non-sensical English sentences consisting of 8 S/KEY-generated words. Each word is chosen from a dictionary of size 2^{10}, which makes the input length 80 bits (same as with the melody generation).

5 Usability Analysis

Armed with two versions of HAPADEP software, we were interested to investigate their respective usability factors. To this end, we performed usability experiments discussed in this section.

We recruited 20 subjects for the experiments. Subjects were chosen on a first-come first-serve basis from the respondents to recruiting posters. Subjects were mainly university students which resulted in a fairly young, well-educated and technology-savvy participant group.

Test Procedure: Our usability tests were conducted in a variety of campus venues (depending mainly on the subjects' preferences), including, but not limited to: cafés, student dorms and apartments, classrooms, office spaces and outdoor terraces. After giving a brief overview of our study goals, participants were asked to fill out the background questionnaire (see Appendix B) to collect demographic information and learn about their music- and computer-related skills and background. Next, users were given a brief introduction to the mobile phones used in the tests to demonstrate some basic operations needed during the test.

Each user was then given the two devices and asked to follow on-screen instructions to complete the given tasks. A user was asked to pair the devices four times in total; twice using the melody variant and twice using the MadLib (sentence) variant. Each variant was tested once with no attack assumption (where verification sequences matched) and once under attack simulation (verification sequences were forced to be randomly different) in order to analyze users' ability

to distinguish matching and different sequences. To reduce the learning effect on test results, the four tasks were presented to the user in random order. The transition between tasks were automated (four executions are started automatically one after the other) and the user actions were logged automatically by the testing framework [16]. After completing the tasks, each participant filled out a post-test questionnaire (see Appendix B) form and was given 5 minutes of free discussion time followed by a short interview.

Results: We collected data in two ways: (1) by timing and logging user interaction, and (2) via questionnaires and structured interviewing.

The sentence-based variant was faster then the melody-based variant when the two values matched. Whereas, the melody-based variant was ahead when the two values differed. Albeit, average completion time hovered around 68 seconds for both methods, as shown in table 1. Although playing a melody takes less time than vocalizing a sentence, the users replayed melodies more to be able to decide whether they were same. When the sentences matched, participants re-played them 1.3 times on average, and 1.75 times when they didn't match. The average play count for melodies was 1.5 for matching and 1.8 for non-matching sequences.

Table 1. Summary of the logged data

Method	Completion Time (sec.)	Fatal Error Rate	Safe Error Rate
Melody (No Attack)	62.15	N/A	10%
Melody (Under Attack)	74.5	15%	N/A
Sentences (No Attack)	56.95	N/A	0%
Sentences (Under Attack)	80.5	5%	N/A

In HAPADEP, if the user indicates (forces) a match in case of two different audio sequences, the protocol clearly cannot be assumed secure. Such a carelessness can allow the attacker to succeed in an impersonation or MiTM attack. Due to its grave effect on security, we call such errors as *fatal errors*. On the other hand, indicating no match for matching values does not introduce any security vulnerabilities but simply voids the current pairing session. We call such errors as *safe errors* due to their *benign* nature. In the context of secure device pairing, notions of *safe* and *fatal errors* are first introduced by Uzun et al. in [28].

As shown in Table 1, using melodies for verification caused higher *fatal* and *safe* error rates. Many participants stated that they would recognize different melodies better if they had a chance to execute the protocol few times and get their hearing "adjusted" before the tests. Subjects who tested the melody variant with non-matching sequences (before matching ones) complained about their initial tendency to tolerate the difference between melodies, since they did not know how much difference they should have expected. Those who tried matching melodies first also complained about the same issue but claimed that they tended to be alerted by slight differences in sequences, due to different quality speakers in the devices. We believe that this is a fundamental problem

with the current melody variant since the security software cannot tolerate any insecure trial-and-error learning period.

On the other hand, comparing sentences resulted in acceptable error rates. There was only one subject who accidentally pressed the *same* button for different sentences causing a *fatal error* to be logged. However, the subject realized his mistake immediately and asked if there was a way to cancel. So, even in this case, the cause of the error was not the user's inability to recognize non-matching sentences but probably our poor GUI design which facilitated this kind of interaction. Security risks of this type of errors can be classified as being lower than those due to unnoticed errors, since the user is aware of the mistake and can thus take an immediate recovery action. However, security software should be free of such errors, since it is hard to accurately foresee the damage an attacker can cause even in few seconds.

In the post-test questionnaire, we solicited user opinions and preference about the tested methods. Participants found comparing sentences easier and more usable in general and preferred to play the sentences one after the other between devices. 95% of the participants rated comparing sentences as easy and 80% preferred it over melodies. Comparing melodies got lower usability rankings, and we observed that it was usable only if both devices started to play the melody at the same time (more-or-less in stereo). Participants were more sensitive to background noise or distractive elements when they were comparing melodies and only 50% of the participants found it easy.

After they filled the post-test questionnaire, we interviewed the participants about their experience with current pairing technologies and HAPADEP. We found out that 70% of the participants tried to setup a secure wireless 802.11 home network and 45% of subjects tried bluetooth pairing before. When we told them HAPADEP (in melody or sentence flavor depending on their choice) can be used as a replacement for those procedures, all people that had previous Wi-Fi pairing experience told they would prefer to use HAPADEP instead. 56% of the people with previous bluetooth pairing experience also said they would prefer HAPEDEP and 22% said they may prefer HAPADEP in certain scenarios but not always. Only four participants had tried infrared communication; two of them said they could not get it to work.

From our usability analysis, we conclude that the HAPADEP melody variant is not mature enough to provide both usability and security. In the rest of the paper, we assume a MadLib (sentence-based) variant.

6 Discussion

In HAPADEP, two devices establish either a unidirectional or bidirectional secure channel by exchanging their public keys over the audio channel. Assuming that the devices are not compromised, the cryptographic primitives and the public key schemes are secure; an attacker can only perform Denial-of-Service (DoS) or Man-in-the-Middle (MiTM) or impersonation attacks.

To perform a DoS attack, the attacker can play a loud audio and prevent the personal device from recording what the target device is playing. In such

cases the transfer phase has to be repeated numerous times. However, such DoS attacks can be recognized (actually, heard) by the user and be traced to its source. The adversary may try to use very low or very high (not audible to the human ear) frequencies but the decoders can be easily tuned to filter out such frequencies forcing the channel always to be human-perceptible.

In an impersonation attack, the attacker's goal is to convey its public key to the personal device by impersonating the target. In the verification phase, however, both devices vocalize sentences that represent their respective views of the exchanged cryptographic material. Note that the target device would compute the sentence based on what it has sent and the personal device computes it based on what it has received. Assuming that the underlying hash function is second pre-image resistant, any impersonation attack would result in different sentences computed on, and vocalized by, the devices. Our usability tests showed that the users are quite capable of recognizing matches and mismatches in device-vocalized MadLib sentences, even in reasonably noisy and crowded environments. Also note that an active impersonation attack would involve a third (adversarial) device attempting to super-impose its sound over one or both legitimate devices and the user can pinpoint the attacker by identifying the third audio source.

While the only requirement for both devices are a speaker and a microphone, the user must be able to perform two (usually) simple tasks:

- Recognize which device is playing the MadLib sentence, among possibly several nearby devices
- Recognize whether two devices are indeed vocalizing the same sentence. (User can replay the sentences as many times as s/he wants and can choose to play them simultaneously or sequentially).

Our usability tests indicated that most non-hearing-impaired adults are capable of performing both activities and so HAPADEP is an easy to use alternative to prior techniques.

Unlike previous proposals, HAPADEP needs only one communication channel. This is crucial when devices don't have any other common communication interface at the time of pairing. If devices have another common interface like Wi-Fi or bluetooth, using HAPADEP is still useful as it dramatically improves the usability by eliminating the need to configure such interfaces. Audio, as a broadcast medium by nature, doesn't need any initial configuration or discovery phase as it is in Wi-Fi or bluetooth. Whenever devices share another common interface and the aim of the pairing is to continue to communicate over it, HAPADEP protocol can be easily modified to transfer the necessary information for automatic configuration of such interfaces. In our prototype implementation, including the bluetooth physical addresses into the exchanged messages needs less than 1 second extra transmission time. After successful termination of HAPADEP, devices are able to automatically initiate secure communication over their bluetooth interface without any user involvement.

The implementation of HAPADEP is straightforward and the source code is available online [2]. We implemented, and experimented with, two variants, based on melodies and sentences (MadLibs). Although the former performed

somewhat poorer than the latter in our experiments, our original motivation for using melodies was two-fold:

- Melodies can be generated on-the-fly without storing any lookup dictionaries.
- Many devices, including those on the low end of the spectrum, are capable of playing chords, but not text-to-speech (TTS).

Storing lookup dictionaries for sentence generation would take up additional storage space of about 50 KB of ROM. TTS engines usually need better computation capabilities and more memory as well. (There are several embedded TTS engines with small footprints that work on almost any cell phone or PDA, but they still do not run on more constrained devices, e.g., bluetooth headsets).

7 Summary and Future Work

This paper introduced HAPADEP – a new approach to secure device pairing. HAPADEP can be viewed as an extension of the previously proposed Loud-and-Clear technique [10] where **all** communication is conducted over the user-perceptible audio channel. HAPADEP is easy to implement and deploy, as our experience indicates. It also offers some built-in protection against DoS and MiTM attacks. The former, in particular, distinguishes it from prior solutions. In addition, HAPADEP doesn't require any common digital interface or initial communication setup and, when available, can help to automate the needed setup of other interfaces thus representing the most usable and lowest-cost device pairing solution to-date.

In HAPADEP, using a different cryptographic authentication protocol may result in smaller footprint and potentially a better usability. Currently, we are working on a new HAPADEP variant that employs a 3-round Short Authentication Strings (SAS) protocol, such as [24,17]. Depending on the availability of a common interface, this variant would need one or three fast codec transmissions and a very short sentence comparison of only two words for verification. The shorter verification phase would decrease the pairing time and the storage needs for dictionaries but the effects of increased rounds and shorter verification sentences on usability is to be addressed in our future work.

Although our usability study clearly indicates that the current melody variant is not the best approach, we are planning to evaluate possible improvements, such as using mixed instruments, different algorithms, forcing simultaneous play of the verification melodies, etc. Making the melody variant a usable alternative is still important as it could be the only option for highly constrained devices that cannot accommodate text-to-speech technology.

Acknowledgements

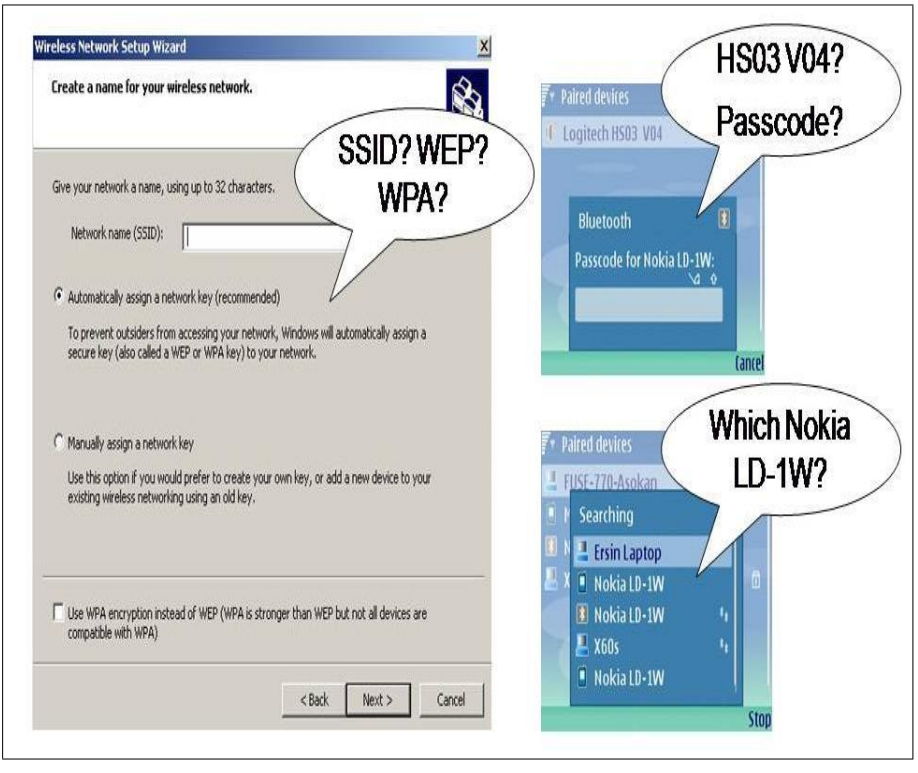
We thank C. Lopes for the help with the audio codecs and Nokia for providing us the phones we used in our tests. We also thank N. Saxena, N.Asokan and the anonymous reviewers for their comments and suggestions.

References

1. Bouncy Castle Crypto APIs, http://www.bouncycastle.org/
2. HAPADEP website, http://sconce.ics.uci.edu/hapadep/
3. Perrig, A., Song, D.: Hash visualization: A new technique to improve real-world security. In: Proceedings of the 1999 International Workshop on Cryptographic Techniques and E-Commerce (CrypTEC 1999), pp. 131–138 (July 1999)
4. Alliance, W.: Wi-fi protected setup specification. WiFi Alliance Document (January 2007)
5. Balfanz, D., Smetters, D.K., Stewart, P., Wong, H.C.: Talking to strangers: Authentication in ad-hoc wireless networks. In: Symposium on Network and Distributed Systems Security (NDSS 2002) (February 2002)
6. Ellison, C.M., Dohrmann, S.: Public-key support for group collaboration. ACM Trans. Inf. Syst. Secur. 6(4), 547–565 (2003)
7. Feeney, L.M., Ahlgren, B., Westerlund, A.: Demonstration abstract: Spontaneous networking for secure collaborative applications in an infrastructureless environment. In: International conference on pervasive computing (pervasive 2002) (2002)
8. Stajano, F., Anderson, R.: The resurrecting duckling: Security issues for ad-hoc wireless networks. In: Security Protocols, 7th International Workshop (1999)
9. Goldberg, I.: Visual Key Fingerprint Code (1996), http://www.cs.berkeley.edu/iang/visprint.c
10. Goodrich, M.T., Sirivianos, M., Solis, J., Tsudik, G., Uzun, E.: Loud and clear: Human-verifiable authentication based on audio. In: ICDCS 2006: Proceedings of the 26th IEEE International Conference on Distributed Computing Systems (2006)
11. B. S. I. Group.Simple pairing whitepaper (2006), http://www.bluetooth.com/Bluetooth/Apply/Technology/Research/Simple_Pairing.htm
12. Holmquist, L.E., Mattern, F., Schiele, B., Alahuhta, P., Beigl, M., Gellersen, H.-W.: Smart-its friends: A technique for users to easily establish connections between smart artefacts. In: UbiComp 2001: Proceedings of the 3rd international conference on Ubiquitous Computing, Atlanta, Georgia, USA, pp. 116–122. Springer, Heidelberg (2001)
13. McCune, J.M., Perrig, A., Reiter, M.K.: Seeing-Is-Believing: Using Camera Phones for Human-Verifiable Authentication. In: 2005 IEEE Symposium on Security and Privacy, pp. 110–124 (2005)
14. Kindberg, T., Zhang, K.: Secure spontaneous device association. In: Dey, A.K., Schmidt, A., McCarthy, J.F. (eds.) UbiComp 2003. LNCS, vol. 2864, pp. 124–131. Springer, Heidelberg (2003)
15. Kindberg, T., Zhang, K.: Validating and securing spontaneous associations between wireless devices. In: Boyd, C., Mao, W. (eds.) ISC 2003. LNCS, vol. 2851, pp. 44–53. Springer, Heidelberg (2003)
16. Kostiainen, K., Uzun, E.: Framework for comparative usability testing of distributed applications, http://sconce.ics.uci.edu/CUF/
17. Laur, S., Nyberg, K.: Efficient mutual data authentication using manually authenticated strings. In: Pointcheval, D., Mu, Y., Chen, K. (eds.) CANS 2006. LNCS, vol. 4301, pp. 90–107. Springer, Heidelberg (2006)
18. Lopes, C.: The *digital voices* project home page, http://www.isr.uci.edu/~lopes/dv/dv.html
19. Lopes, C.V., Aguiar, P.M.: Acoustic modems for ubiquitous computing. IEEE Pervasive Computing 02(3), 62–71 (2003)

20. Lopes, P., Aguiar, C.V.: Aerial acoustic communications. In: 2001 IEEE Workshop on the Applications of Signal Processing to Audio and Acoustics, pp. 219–222 (2001)
21. Mayrhofer, R., Gellersen, H.: Shake well before use: Authentication based on accelerometer data. In: Proc. Pervasive 2007: 5th International Conference on Pervasive Computing (2007)
22. Microsoft. Windows connect now-ufd and windows vista specification. version 1.0 (2006), http://www.microsoft.com/whdc/Rally/WCN-UFDVistaspec.mspx
23. Saxena, N., Ekberg, J.-E., Kostiainen, K., Asokan, N.: Secure Device Pairing based on a Visual Channel. In: 2006 IEEE Symposium on Security and Privacy (2006)
24. Pasini, S., Vaudenay, S.: Sas-based authenticated key agreement. In: Yung, M., Dodis, Y., Kiayias, A., Malkin, T. (eds.) PKC 2006. LNCS, vol. 3958, pp. 395–409. Springer, Heidelberg (2006)
25. Prasad, R., Saxena, N.: Efficient device pairing using human-comparable synchronized audiovisual patterns. In: Applied Cryptography and Network Security (ACNS) (June 2008)
26. Roth, V., Polak, W., Rieffel, E.G., Turner, T.: Simple and effective defense against evil twin access points. In: WISEC, short paper, pp. 220–235 (2008)
27. Soriente, C., Tsudik, G., Uzun, E.: BEDA: Button-Enabled Device Association. In: IWSSI (2007)
28. Uzun, E., Karvonen, K., Asokan, N.: Usability Analysis of Secure Pairing Methods. In: Dietrich, S., Dhamija, R. (eds.) USEC 2007. LNCS, vol. 4886. Springer, Heidelberg (2007)
29. Diffie, W., Hellman, M.E.: New directions in cryptography. IEEE Transactions on Information Theory IT–22(6), 644–654 (1976)
30. Wireless USB Specification. Association models supplement. revision 1.0. USB Implementers Forum (2006), http://www.usb.org/developers/wusb/

Appendix A: Screen-Shots of Some Confusing Steps in Wi-Fi and Bluetooth Connection Set-Up



Appendix B: Background and Post-test Questionnaires

Background Questionnaire

Demographics

Age

☐ 18-24 ☐ 25-29 ☐ 30-34 ☐ 35-39 ☐ 40+

Sex

☐ Male ☐ Female

Highest Grade Competed

☐ High School ☐ Bachelor ☐ Masters ☐ Doctorate

Computer experience

For how long you have been using computers?

On a typical day, how many hours do you work with computers?

Professional/Amateur music related experience

In a normal day, for how many hours you listen to the music?

☐ <1 ☐ 1-3 ☐ 3-5 ☐ >5

Do you play any musical instrument?

☐ YES

For how many years?

Do you consider yourself an amateur or a professional?

☐ Amateur ☐ Professional

☐ NO

Have you participated in any professional music related activity?

☐ YES (your title/position was ……)

☐ NO

Posttest questionnaire

Please answer the following questions based on your experience using the method. Where appropriate, we would appreciate if you would explain your answers and reasoning in the spaces provided or orally to us.

1. I found the method that requires comparison of melodies
   - ☐ very easy to use
   - ☐ easy to use
   - ☐ hard to use
   - ☐ not usable at all

2. I found the method that requires comparison of sentences
   - ☐ very easy to use
   - ☐ easy to use
   - ☐ hard to use
   - ☐ not usable at all

3. It easier to identify whether the verification melodies are same/different; when I play the melodies on the devices
   - ☐ At the same time
   - ☐ One after the other

4. It easier to identify whether the verification sentences are same/different; when I play the sentences on the devices
   - ☐ At the same time
   - ☐ One after the other

5. Please choose the method that you would prefer to use
   ☐ Comparing Melodies ☐ Comparing Sentences

6. I would prefer to hear something else (such as mix of instruments, animal sounds, etc.) instead of a melody/sentence to compare.
   - ☐ YES (I would prefer ……)
   - ☐ NO

7. Please add any comments in the space provided that you feel will help us to evaluate the method or come up with a better one. (You can answer this question orally if you would like to).

One-Time Password Access to Any Server without Changing the Server

Dinei Florêncio and Cormac Herley

Microsoft Research, One Microsoft Way, Redmond, WA
dinei@microsoft.com, c.herley@ieee.org

Abstract. In this paper we describe a service that allows users one-time password access to any web account, without any change to the server, without changing anything on the client, and without storing user credentials in-the-cloud. The user pre-encrypts his password using an assigned set of keys and these encryptions are sent as one-time passwords to his cell phone or carried. To login he merely enters one of the encryptions as prompted, and the URRSA service decrypts before forwarding to the login server. Since credentials are not stored (the service merely decrypts and forwards) it has no need to authenticate users. Thus, while the user must trust the service, there are no additional passwords or secrets to remember. Since our system requires no server changes it can be used on a trust-appropriate basis: the user can login normally from trusted machines, but when roaming use one-time passwords. No installation of any software or alteration of any settings is required at the untrusted machine: the user merely requires access to a browser address bar.

Keywords: Passwords, one-time passwords, authentication, replay resistance.

1 Introduction

Users increasingly find themselves in the position of having to enter sensitive information on untrusted machines. As access to more and more services is pushed online, the range of sensitive information that a user must protect grows with time. Passwords are the most obvious example. Email, bank and brokerage accounts, employee benefits sites, dating and social networking sites almost universally allow password protected access to services. A user who logs in to any such account from an untrusted machine runs the risk that a keylogger will capture the password and allow unauthorized access. In addition, the number of machines that must be regarded as untrusted also grows. Most obviously, any machine at an internet café or kiosk must be assumed suspect. But additionally a user's own home computer can easily be infected with spyware.

The problem we address is to enable a user to login from a machine that is untrusted. For simplicity we will assume the worst: everything the user does on such a machine is observed and logged. Everything typed, everything that appears on the screen, and all of the network traffic is captured and is available

T.-C. Wu et al. (Eds.): ISC 2008, LNCS 5222, pp. 401–420, 2008.

to an attacker. Nonetheless we want to be able to login to password protected accounts from such a machine, without risking catastrophic loss of data. While there are widely varying estimates of the dollar size of the fraud problem that password stealing causes [20] the fear and confusion is very real. We assume that preventing passwords from falling into the wrong hands is more important than protecting the rest of the data from a session. Thus we are not protecting the privacy of data, and we do not prevent session hijacking.

Our approach for passwords to a particular account will be to generate a series of one-time passwords that can be used to login. The actual mechanism we employ will be for the user to navigate to a webserver that will act as a Man-In-The-Middle (MITM). We call the system URRSA: Universal Replay-Resistant Secure Authentication. The one-time data will be provided to the URRSA server, which then performs a decryption and substitution: replacing the one-time password typed at the suspect machine with the true password forwarded to the login server. In this way the sensitive information is not typed at the untrusted machine, and neither is it displayed or downloaded to the compromised environment; nor is it stored at the URRSA server.

We make several requirements of the solution in order to be useful:

- No change to existing login server.
- No change to the browser or client software environment. We do not assume that the roaming user has installation privileges. We do not require the user to change the browser proxy settings. Requiring users to alter browser proxy settings we believe makes the User Experience of Impostor [27] and KLASSP [15] unworkable for a realistic deployment.
- No storage of credentials in the cloud: this removes the single point of attack that such a server would represent.
- No authentication of the user to the service: if we try to authenticate with a password we are back where we started. The alternatives, such as smartcards, greatly increase the complexity of the service and the burden on the user.

Our main goal in this paper is to describe the technology and give sufficient detail to allow implementation. However, it is legitimate to question whether users will trust such a system. The answer is obviously dependent on who runs it. There are two deployment scenarios that don't involve trusting a third party. The first is that the login server runs the service. For example, using URRSA, PayPal or Fidelity might offer OTP access to those clients who desire it *without altering their current authentication process.* The second is that the user self-hosts the service on a machine that he controls. Obviously this solution requires that the user maintain a server and a fixed IP address or a domain name; so this is possible only for a small minority of users. Both of these deployment models are potentially useful, and get around the issue of trust by having the "third party" be one of the existing two parties. The final model, and potentially most useful, is of URRSA offered as a web-service hosted by a third party. The success of online financial management sites such as `www.yodlee.com`, `www.mint.com` and `www.wesabe.com` demonstrates that at least some users will trust such a service.

At `Yodlee`, for example, users give the service passwords to their bank and credit-card accounts so that it can daily update bill and payment information.

In the next section we review related work. In Section 3 we show how any account can be transformed into a one-time password account without having to change the server or the browser. We review implementation details in Section 4. Section 5 examines attacks and Section 6 evaluates our deployment.

2 Related Work

2.1 Coping Strategies and Simple Tricks to Evade Spyware

Sometimes coping strategies can be enough to evade a keylogger. Herley and Florêncio [11] describe a simple trick that users can employ to confound keyloggers by obfuscating their passwords. By interspersing the legitimate password characters with random characters typed outside the password field, the technique is able to confuse most existing keyloggers. While useful, this is not a durable solution, as keyloggers could be easily modified to capture enough additional information to retrieve the actual password.

Another technique that can be used to authenticate without explicitly typing passwords involves storing the password or equivalent information in a bookmark, and accessing it from a standard web browser. A flexible way of achieving that is based on the use of *bookmarklets*. These are small JavaScript programs that are stored as bookmarks. JavaScript is flexible enough that it can be even used to hash a general password, and generate site-specific passwords [1]. Storing the password (or equivalent information) in a URL link, bookmarklets avoid the need to type the target site password. Access could be achieved when roaming, by storing the bookmarklets on a USB drive, for example. Nevertheless, this is not necessarily safe. If its use were to become widespread, hackers could attack the bookmarklets directly. If the file containing the bookmarklets is copied when inserting the USB drive, every single password on the drive could be compromised (thus making the user less secure not more). It would make matters worse not better if checking a `hotmail` account exposed the `BankOfAmerica` credentials stored on the same device to be exposed.

Another group of solutions involves having the web site use some authentication method other than simply typing passwords - sometimes in combination with some typing. Examples include on-screen keyboards, two-factor authentication, challenge-responses systems, and many others. These usually apply only to the site that adopts that particular mechanism, and require a major change in the server and User Experience. Rather than have users key their passwords some web sites have experimented with on-screen keyboards as a method of secure data entry. These schemes can be attacked by having the keylogger do a screen capture at each mouse click event. An interesting work by Tan *et al.* [31] addresses the question of minimizing the chances that a password entered using an on-screen keyboard is captured by an observer. This work addresses the "shoulder surfing" risk rather than the risk that the machine itself is

running spyware, but has interesting analysis of the usability of various alternative password entering mechanisms.

2.2 Challenge Response Mechanisms

The use of challenge response has been explored as a means of achieving resistance against replay attacks when the user must login from an untrusted environment. Cheswick [12] examines on a higher level the use of Challenge-Response authentication mechanisms to evade spyware. The advantage of such systems is that a spy who observes a successful login session cannot perform a replay attack: the challenge will be different for each event and observing a single response helps the attacker very little. Cheswick reviews a number of approaches from the point of view of usability. Each of these schemes would require changes at the server.

Pering *et al.* [28] explore the use of a series of the user's own images uploaded in advance. The user is then authenticated by successfully responding to a series of challenges, which essentially involve picking his images from random images. Another image based scheme is proposed by Weinshall [32]. To avoid an image-similarity attack, the images are assigned, rather than uploaded. Furthermore, to reduce the amount of information given out with each authentication session, the user is not directly asked which images are his. Instead, the image set memberships are used to select a certain path on an image mosaic, with the user providing only a code that depends on the path's endpoint. It is pointed out by Golle and Wagner [18] that observing as few as six logins of this scheme can allow an attacker to determine the secret. Coskun and Herley [13] show that a challenge response scheme that relies on the users memory and calculating ability alone is almost certainly vulnerable to brute-force attack.

2.3 Proxy-Based Systems

Four works that directly address authentication from untrusted machines are Impostor [27], that of Wu *et al.* [25], Delegate [21], and KLASSP [15]. All use a proxy to intervene.

The Impostor [27] system of Pashalidis and Mitchell, is a password management system where roaming users can access their credentials. Rather than have users authenticate themselves by typing a master password (as is the case for [17]), a challenge response authentication is used. The user is assigned a large string that forms the secret. When requesting access the user is challenged to provide characters from randomly selected positions in the string, and is authenticated only if she responds correctly. In this way the user reveals only a small portion of the secret string to any compromised machine. A replay attack is difficult, since the challenge positions change each time the user contacts the proxy. Nonetheless, Impostor potentially protects strong secrets with weak ones. If strong (*e.g.* 60 bit) passwords are stored with the system an extension of the three character challenge originally proposed would be necessary (as shown in [13] such a scheme is vulnerable to an attacker who observes several logins). Impostor runs as a HTTP proxy,

and the user must direct their browser to the proxy. Wu *et al.* [25] sketch a similar architecture where a proxy stores credentials; the proxy delivers a challenge which must be answered by SMS to authenticate the user.

Another proxy-based system which stores users credentials is Delegate of Jammalamadaka *et al.* [21]. Like Impostor, they store the passwords in the cloud, and act as a proxy to serve as intermediary between the server and the untrusted terminal. Credentials are filled by the proxy in web requests as they are forwarded to the login server. A cell phone is used for the user to explicitly authorize credential insertion when necessary. This requires, of course, that the user has cell reception. By contrast our system has no such requirement. An additional feature is that Delegate uses a rule-based hierarchy to request additional authentication whenever a sensitive operation is requested. This can be used to reduce the risks of a session hijacking (e.g., by requesting additional authentication when a money transfer is requested) as well as to remove sensitive information (e.g., account balances) from web pages provided by the server. These rules are generally to be provided by security experts, or learned from the user on a previous interactive session from a safe terminal.

The KLASSP proxy [15] of Florêncio and Herley also functions as a MITM proxy for communication with the login server. The user enters a *mapped* password $M(pwd)$ on the untrusted machine, and the proxy *unmaps* before forwarding $M^{-1}M(pwd)$ to the login server. The mapping $M()$ has, of course, been agreed in advance between the user and the proxy and serves as a shared secret. KLASSP suggests two broad directions for mappings. In the first the user obfuscates the password by entering either password keys or random keys in response to prompts from the proxy (in a variation on [28] the prompts are a series of images, where the user's personal images act as sentinels to signal a true password character). As with Impostor, this technique protects strong secrets with weak ones. In the second the user encrypts the password using a large and cumbersome encryption table.

Impostor and KLASSP have in common that they are implemented as HTTP proxies. This means that the user must force the browser on the untrusted machine to use the proxy. This is inconvenient and is not always possible (*e.g.* the user may not have permissions). In Internet Explorer this requires editing Tools, Internet Options, Connections, LAN settings, un-checking "Automatically detect settings", checking "Use a proxy server", entering the IP address and port number, clicking "Advanced options" and checking "Apply same proxy for all protocols." Further the settings must be undone when the user leaves; if this is neglected or forgotten there is a risk that the next user of the machine has his traffic routed through the proxy also.

2.4 One-Time Passwords and S/Key

Several one-time password systems have been proposed that limit the attacker's ability to exploit any information he obtains. Notable among them is S/Key [22,29] which generates a series of passwords by iteratively taking a cryptographic hash of a secret key. At each login the server verifies that the hash of what the

user presents is the previously used password. Since each of the passwords is used only once it is of no use to an attacker. A further advantage is that the server need store only the previously used password. Thus even the database at the server contains nothing useful to the attacker. A popular implementation of S/Key for Unix is described by Haller [19]. The user carries a list of OTP's; generally these are sequences of short English words, so the user must type a total of about 20-24 characters to authenticate. An alternative one-time password system is OTPW by Kuhn [2]. Instead of being generated from a single secret (as with S/Key) here the passwords are independently chosen random secrets, and the hash of each is stored on the server.

Mannan and van Oorschot [24] describe MP-Auth: a system that uses a trusted mobile device such as a PDA or smart-phone to enter the password. The device encrypts the password using the end server's public key before passing it to the untrusted terminal. MP-Auth has the advantage of not requiring (as we do) that the user trust a proxy, but does not work with existing login servers, and requires a channel (such as bluetooth) between the trusted device the untrusted machine.

SecurID from RSA [3] gives a user a password that evolves over time, so that each password has a lifetime of only a minute or so. This solution requires that the user be issued with a physical device that generates the password. This solution requires considerable infrastructure change on the server side, which has limited its use. However SecurID has the advantage of being immune to OTP stealing attacks (see Section 5.3).

2.5 In-the-Cloud Password Managers

One sub category of challenge response system is worth a separate note: in-the-cloud password management systems. These systems store the sensitive information at a server in-the-cloud, and have the server deliver the sensitive information directly to the desired destination on the user's behalf. An early example is [17]. Storing all this sensitive information in the server provides a new vulnerability. Indeed, if an attacker gains access to the user's account at this server, it would have access not only to the information that the user typed, but to any other information stored there as well. Further, a server storing hundreds or thousands of users sensitive information can itself become a target for attacks.

An early in-the-cloud example, proposed by Gaber *et al.* [17], used a master password when a browser session was initiated to access a web proxy, and unique domain-specific passwords were used for other web sites. Since users authenticated themselves by typing the master password, this clearly offers no defence against keyloggers. The same is true of other in-the-cloud systems such as Passport, where the user authenticates himself using a master password, or `www.clipperz.com` where a passphrase is used.

2.6 Relation to Our Service

One-time Passwords offer well understood security enhancements over existing password systems. Our proposed scheme gets the excellent protection enjoyed by

users of existing OTP systems [22,19,2,3] to all users. We wish to be clear that we will have the same security and usability questions that arise with other OTP systems; *e.g.* from a usability standpoint the user must keep the OTP list safely, and, like most OTP systems, session hijacking is still possible. The advantages we offer over previous approaches is that we give OTP protection *without changes to the existing server*. A consequence is that users can have trust-appropriate authentication. When using a trusted machine they can continue to login as before. However when they decide the circumstances demand they can use one-time passwords to access the account. When an OTP list is generated it is sent by SMS text message to the user's cellphone, but users who prefer may print and carry a hardcopy OTP list. We view the cellphone as preferable for a number of reasons. It represents a device that the user generally carries anyway. The user is less likely to lose or misplace a cellphone than a hardcopy OTP list. Finally, by using an out of band channel like SMS to send OTP's to the user new OTP lists can be generated without requiring the user to return to a trusted location (see Section 4.3).

An advantage of the system we propose with respect to the proxy-based systems Impostor and KLASSP [27,15] is that it is implemented as a MITM web service rather than a HTTP proxy. Users do not need to change the proxy settings on the browser before they begin and undo them when they are done. Thus the burden is much lower than with [15] or [27]. Further the service is involved only for the duration of the connection to a password protected account. For example an Impostor or KLASSP user during a one hour session might change the proxy settings at the beginning and undo them at the end. If he visited several password protected accounts, but also news and information sites the proxy would handle all of the traffic for the entire hour. With our MITM service implementation the URRSA server is involved only from login to logout on each password protected account. The entire traffic for `BankOfAmerica` would flow through the service, but none of the general browsing traffic would. Thus the load on the service is greatly reduced (in comparison with Impostor or KLASSP) and privacy is enhanced. Delegate [21] does not explain how its proxy mechanism works. We assume, since it makes no mention of the crucial processing of the request-response stream that is the heart of our system (Section 4.2), that it is also implemented as a HTTP proxy.

The URRSA service can be seen as a descendant of KLASSP [15], where the mapping $M()$ becomes a true encryption of the password, and the proxy is a reverse proxy. Independently, and after, one of the authors of Impostor also developed and deployed a similar system [4]. This appears to be based on a reverse proxy similar to CGIProxy rather than the scheme we describe in Section 4 but is similar in many other respects to our service.

3 Method

URRSA provides a service that allows users to access a website requiring authentication, without having to type the actual password in the clear. The only time the actual password is typed is during the registration, which is done in

advance from a safe location. At registration, the user receives versions of the true password, each encrypted with a different key. The service will decrypt using any of the keys only once, so effectively the user receives One Time Passwords each of which can be used once to access the registered site from untrusted locations. Our approach does not store any passwords at the URRSA server. The server needs to store only the encryption keys used, not the actual password. By doing this we remove the information stored with the service as a vulnerability: the keys have no value to the attacker without the corresponding OTPs (and if an attacker has the OTPs, he doesn't need to steal the keys, just use the service). More importantly, we remove the need to authenticate the user. We wish to be clear however that, while the service does not store user passwords, the user must still trust the service.

3.1 Mapping Strategy

In theory passwords can contain upper and lower case letters, digits and any of a few dozen special characters. While in practice we know that the majority of users seldom use extended characters [14] we must nonetheless support all possibilities. Letters and digits give 62 characters and we allow for 66 special characters, for a total of 128 possible characters. Call this set $\mathbf{C}$. An obvious way of encoding the password characters would be to use a simple permutation code that maps $\mathbf{C}$ to itself, and to apply this to all characters independently. Thus a length N password (*i.e.* in $\mathbf{C}^N$) would be mapped to another password in $\mathbf{C}^N$. There are a few problems with this approach. First, permutation codes leak information when two characters of the original password are the same. More significantly we have the following complications:

- Confusion sets: certain characters such as the number "zero" and the letter "O", or the number "one", the upper case "I" and lower case "L" can be hard to tell apart when context is removed.
- SMS restrictions: certain characters (*e.g.* []{}) cannot be sent by SMS (see *e.g.* [5]).
- Unfamiliar keyboards: layout for the position of special characters varies greatly on international keyboards. Some characters requiring meta-keys (*e.g.* Shift, Alt, Alt-Gr *etc*) can be very hard to find.

It is important to exclude confusion sets. This is especially the case if the user carries the OTP list on his cell phone, since we have no control whatever over the fonts in which the OTP will be displayed. We require that every password map to an OTP that is unambiguously readable on any display. This rules out "O" and "0" *etc*. The set of characters that cannot be sent by SMS must be excluded from any mapping we produce. Since the phone will merely receive and display we have no opportunity to perform any mappings there. Finally, even common special characters such as "@" can be hard to find on an unfamiliar keyboard (*e.g.* on many keyboards it requires pressing the Alt-Gr key which often causes confusion since Alt-Gr does not exist on US keyboards). The problem is compounded when

the meta keys such as SHIFT, Alt, Alt-Gr, Esc *etc* are labeled in a language or alphabet unfamiliar to the user. While we wish to support the minority of users who have unusual characters in their passwords we do not wish to force a user with a simple password such as "Snoopy2" to search for characters like "%" and "¿" on the keyboard in a Chinese internet café.

To avoid any and all confusion, we restrict the output OTPs to use only capital letters and digits, not including the above mentioned characters: "0", "O", "I", and "1". Therefore the valid characters are easily identifiable: ABCDEFGHJKLMNPQRSTUVWXYZ23456789. Call this set **D**. This gives us 32 characters, enough to carry 5 bits of information per character.

So we have an input password with N characters drawn from an alphabet of 128 symbols (*i.e.* the set **C**). We wish to map to an OTP drawn from an alphabet of 32 symbols (*i.e.* the set **D**). Clearly the OTP must be longer than the input password. We transform the input password to a string of $7N$ bits, and encrypt those bits using the one time encryption pad. We then map the result (5-bits at time) to a password OTPi with M symbols drawn from an alphabet of only 32 symbols. Clearly, OTPi will have $M = ceil(7N/5)$ characters. Thus the procedure maps a password from $\mathbf{C}^N$ to an OTP from $\mathbf{D}^M$. For example the 9 character input password "{Qp#oL{4s" might map to the 13 character OTP "RM8BQ47AAKW3U." The OTP contains only characters that are unambiguously readable on any display and easily found on any keyboard. We then repeat the process with different encryption keys to produce each of the desired OTPs. For decoding, we follow the reverse procedure. Pseudo-code for this encoding is given in Section A.1. To decrypt the password, the URRSA server needs only to know which encryption key was used. The url and userID pair allows this to be determined. After receiving this information, the server checks which was the last key used and informs the user which OTP to use next. This information is all that is required to tell the Service which key to use (see Section 4.2), and to guarantee that service and User are in sync. Since the service merely decrypts and forwards no authentication of the user with the service is required.

As described each input password of a length N would map to an OTP of length $M = ceil(7N/5)$. However, since we know that a majority of users choose weak passwords [14] this is actually somewhat wasteful. The two passwords "snoopy" and "G(r!e9" will map to the same length. If we Huffman encode [30] the plaintext password before encryption and mapping we can reduce the length of the average OTP that must be typed. Huffman decoding is done at login time. Note that we have not weakened user passwords in so doing; we have merely ensured that weak passwords from $\mathbf{C}^N$ will be mapped to the shortest possible strong password. It is also worth noting that password strength does not necessarily increase security when password stealing attacks such as phishing and keylogging are the main threats [16].

3.2 User Experience

The user merely navigates to `http://{URRSA}/OTPLogin` login page at the webserver and enters first the url and userID of the account he wishes to access

(this allows the proxy to retrieve the keys used to generate that user's one-time passwords). The user then enters the k-th OTP from his OTP list, allowing the server to decrypt and temporarily store the true password. The user's browser is directed to open `https://{URRSA_1}` (which directs our server to fetch the registered login page). The user need type nothing further and merely clicks the submit button and login proceeds. Observe that the URRSA service does not authenticate the user. It has no need to, since it does not know any of the user's passwords and merely decrypts and forwards.

If we compare the user experience between logging in directly (*i.e.* navigating directly to the login server and risking a keylogger) and using our service we find as follows (we will use PayPal as an example). To go directly to the login server the user types `https://www.paypal.com` in the address bar and then his userID and password and clicks submit. To login using our service the user types `http://{URRSA}/OTPLogin` in the address bar, then types `www.paypal.com` and his userID at the loaded page and submits. A new page loads on which he enters the requested OTP and submits. Finally, when the `https://{URRSA_1}` page loads (which will be a copy of the `https://www.paypal.com` just loaded by the service) he clicks submit one more time. Thus it can be seen that the additional burden on the user is not very great: the user has one additional URL to type, and two additional clicks. The sequence of events is detailed in Section A.2.

3.3 Acting as a MITM Webservice

The MITM service that URRSA performs can be regarded as a reverse proxy [23]. Conceptually (if we dealt only with static HTML pages) a reverse proxy works by fetching a first document for the client and translating all links therein to again go through the proxy. For example if the client wants `https://www.abc.com/foo` it would instead ask for `https://{URRSA_1}/foo`. The proxy receives the request and forwards to the server at `www.abc.com`; the server delivers the request to the proxy, which passes it back to the client browser. This is not to be confused with a HTTP proxy, where the browser is configured to direct all traffic to the proxy [23]. Reverse proxies are also sometimes known as CGIproxies after a particularly popular family of implementations [6]. While conceptually simple, reverse proxying modern web-sites is a complex task. Examples of reverse proxy can be seen at any of a number of anonymizing web proxies. However most anonymizing proxies offer a somewhat brittle experience [26]. In particular dynamically generated links are often handled incorrectly, and missing images and broken links are a common experience. In addition many are unable to handle SSL traffic successfully, certificate errors are common, and cookies do not always get correctly assigned. We are unaware of a single anonymizing reverse proxy that is robust enough to handle the complicated request/response stream that occurs during an authentication.

We are able to simplify the problem by attempting considerable less generality than anonymizing reverse proxies offer. Rather than reverse proxy for any possible domain, and all the links therein we seek to handle only the limited number of domains and sub-domains encountered during login at a site. A login

server generally has links to fewer external domains than a conventional site (*e.g.* when logging into `www.PayPal.com` a user sees essentially only content from the `PayPal` and `PayPalObjects` domains while loading `www.nytimes.com` involves as many as ten distinct domains). So our task will be to translate only for the domain(s) to which the user is authenticating and not for a plurality of sites.

4 Implementation

We have implemented the URRSA server and deployed on an internet facing machine: see Section 6 below. We now address some of the issues related to implementation. This architecture and flow of events during an authentication is illustrated in Figure 1. We use ASP.Net scripting to handle the the actions to be performed at the web server. There are three web services running on the server: `OTPRegistration`, `OTPLogin` and `OTPRefresh`, which we review in turn.

4.1 Registration Webservice

To use the service a user must first navigate to `{URRSA}/OTPRegistration` and get a list of one-time passwords. He enters the url and userID of an account he wishes to access and is assigned a randomly chosen set of keys. Recall that the url and userID pair uniquely identifies the user, and hence the set of keys issued. He also enters the password, *pwd*, for this account; and indicates the cellphone number he wishes the list sent to. The webservice then generates a list of 10 one-time passwords and sends them by SMS text message to the desired number.

For this step `OTPRegistration` interfaces to an SMS gateway service. There are several providers which expose programmable interface to make sending text messages simple. In our implementation we use Clickatell [7] which offers a variety of means of triggering the send message. In the simplest, after establishing an account and paying for credits, a message can be sent merely by navigating to: `http://api.clickatell.com/http/sendmsg?user=xxxxx&password=xxxxx&api_id=xxxxx&to=xxxxxxx&text=xxxxxx`, where the fields to be filled are the user account name, password, application id, destination phone number and SMS message. This could be invoked by causing the user's browser to navigate to the appropriate address once the OTP list has been calculated. While simple, this leaves our password to the SMS gateway (though not the user's password) in the clear on the trusted machine. While the machine is trusted by the user, the user is not necessarily trusted by the server, and this would potentially allow him to replay and exhaust the message budget at the gateway. Instead we use an API integrated with the server, which also causes the desired message to be sent. We are limited to only 10 passwords by the 160 character limit that applies to SMS messages. This step must be done at a trusted machine. The keys are stored at the server along with the url and userID, but no record of the password is kept.

In the event that the user does not have a cellphone he may elect to carry a copy on paper. In this case the webservice then generates a larger list of 30

one-time passwords which he prints and carries. The OTP's can be carried on a PDA, or an mp3 player that is capable of displaying text files. This has the advantage that storage is no longer an issue. As with any OTP system the user must protect the list (see coverage of attacks in Section 5).

4.2 Login Webservice

Now to login the user navigates to `{URRSA}/OTPLogin`. He is asked for the url and userID of the account he wishes to access. Since this pair uniquely identifies him this allows the proxy to retrieve the keys. For the k-th login the user is prompted to enter the k-th OTP from his list: OTPk. The server decrypts to get the true password. The user's browser is instructed to automatically open `https://{URRSA_1}`. Using the `onclick` event for the "Submit" button we can use, for example the Javascript `Open()` command, which causes a new window to open with a specified URL.

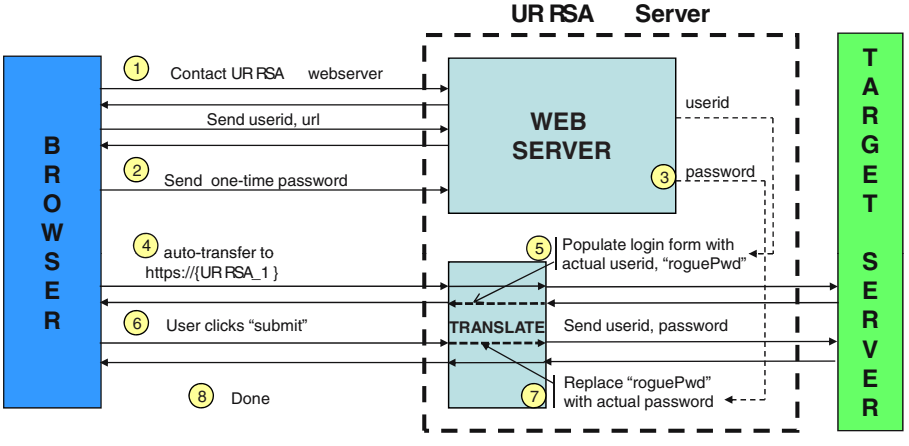


Fig. 1. The sequence of steps logging in using the URRSA service. See a description in Section A.2. The heart of the service is the translation which acts as a MITM service: it sits between the client browser and the login server and edits the request/response traffic between them. The implementation this part is described in detail in Section 4.2.

Backend Processing. To reliably translate as a MITM service between `https://{URRSA_1}` and `https://www.paypal.com` our URRSA implementation must perform [23]:

- request URL mapping
- request header mapping
- response header mapping
- response body link translations
- cookie re-assignment
- certificate replacement.

These changes can be implemented directly on the request/response stream using the `WebClient` and `HttpWebResponse` classes in .Net. There are also a variety of proxy applications that offer the ability to modify in real-time request/response traffic [8,9,10]. The most common mapping that must be performed on the request header is to the `Host` field. Requests that are constructed relative to a given host will require mapping of this field. For example, in requesting `https://www.BankOfAmerica.com/accounts` the GET request issued by the browser is for `/accounts` and the `Host` from which it is to be retrieved is in the header. However, when using a reverse proxy we want the browser to request `/accounts` from `URRSA_1`. The proxy must then map `URRSA_1` to `www.BankOfAmerica.com` as the request goes by.

Response headers often contain information about the document in the `Location` field. The most common case is to indicate the new location when a document has moved. This is a very common way to allowing sites to add and change web site content and point several access points at a single login page. Before forwarding the response back to the browser we have the proxy map `www.paypal.com` to `URRSA_1`. Cookies play an important rôle at many login servers. When a cookie is set by PayPal the same origin policy will cause the browser to return that cookie only with requests sent to PayPal. This is problematic, since the browser at the untrusted machine is connected to `URRSA_1` rather than `www.paypal.com`. We solve this by re-assigning the domain to which cookies belong in the response header.

When SSL connected to PayPal the browser receives a PayPal certificate signed by Verisign (a Certificate Authority (CA) trusted by most browsers). The URRSA server will maintain two SSL connections. From PayPal it receives a PayPal certificate signed by Verisign, just as a regular client would. For the SSL connection it maintains with the client it must have its own certificate, also signed by a CA trusted by the browser. Thus the user will never see a PayPal certificate, but rather one for `URRSA_1`.

The response content must have all references to the end server replaced with ones to the proxy. For example translate requests such as `https://www.paypal.com/images/logo.gif` to `https://{URRSA_1}/images/logo.gif`. Table 2 lists the rules to carry out the mappings referred to above.

The changes described so far allow us to translate for a single host. However, for many sites content is loaded from more than one host. For example, when logging into PayPal the browser loads content both from `www.paypal.com` and `www.paypalobjects.com`. For gmail content is loaded from `mail.google.com` and `www.google.com`, while for sites such as myspace as many as a dozen or more hosts can be involved. We solve this by having a single IP address, or host, at the proxy handle each host that is involved in a login at the end server. For example for Paypal the translations for `www.paypal.com` will be handled at `URRSA_1` while those for `www.paypalobjects.com` will be handled at `URRSA_2`. This has the major advantage that relative links in the response are resolved automatically without intervention by the proxy. This represents a major point of difference with reverse proxies based on CGIProxy: these use a single proxy host for any and all hosts

involved in the login session. Thus all relative links must be found and translated. A detailed description of reverse proxying can be found in [26].

In addition, recall that we must insert the user's decrypted password into the request as the login page is submitted to the server. When a login page is loaded the response content (Step 5 of Figure 1) contains a password field. We auto-populate this field by replacing in the response body the string `type=''password''` with `type=''password'' value=''roguePwd''`. This causes the login page that appears to the user to have an already filled in password field (this is not of course the user's password, but rather the "roguePwd" string). We have a further rule that replaces in the request header the string "roguePwd" with the just decrypted user password (Step 6 of Figure 1).

Having the login page appear with an auto-populated password field serves a number of functions. First, it provides the sentinel value for which the URRSA server will search the request header to do the actual password switch. Second, many login pages will not allow submission with an empty password field, so the field must contain something. Finally, in having the field auto-filled with a value that is of no use to an attacker we greatly reduce the risk that the user reflexively types his password when faced with an empty login page.

Table 2 lists the rules to carry out the mappings between `www.paypal.com` and `URRSA_1` referred to above. A similar set of rules would translate between `www.paypalobjects.com` and `URRSA_2`. Essentially any server can be handled in this way, where we dedicate an IP address or host at the proxy to each host at the end server. A more scalable reverse proxy scheme is described in [26].

4.3 Refresh Webservice

When the user requires a new OTP he can re-register. Alternatively, he can have a new list generated and sent to his cell phone. This works as follows. A service `{URRSA}/OTPRefresh` resembles the `{URRSA}/OTPRegistration` service. The user identifies himself by giving the url and userID of the account for which he requires a new OTP list. However, instead of presenting his true password for encryption he submits the last OTP on his list. This allows the server to retrieve the key i. The proxy decrypts to get the true password, and then re-encrypts to form a new list and transmits them to the desired number. In this manner the user can repeatedly refresh his OTP list without having to return to a trusted machine. Of course, he must refresh the list before he uses the last OTP on his list. OTP Refresh is not available if the user carries the list by paper, since the proxy has no out-of-band channel to send a fresh list.

5 Attacks

5.1 Lost or Stolen OTP List

First, the technology is a one-time password (OTP) technique, and therefore, subject to the same kind of vulnerabilities as other OTP systems. It deserves emphasis that the list must be generated at a trusted location and should be kept carefully.

The OTP list is sent by SMS text message to the user's cellphone, but printing and carrying a hardcopy is also supported. The list contains the url of the login server along with the OTP list, but not the userID. If the OTP sheet is lost or stolen the finder will possess a series of one-time passwords, but will not know the userID of the account for which they work. We recognize that this is an imperfect defence, and do not claim to have solved the problem of users who are careless or lose their OTP sheet (it is for this reason that we regard the phone as a better channel). However a user who discovers he has lost his OTP sheet can render it useless by generating a new sheet. If he can go to a trusted PC he generates a new sheet and the old one is worthless (since a new set of keys is generated). A user who cannot reach a trusted PC can still render the lost OTP sheet worthless by re-registering at an untrusted PC. By typing random characters instead of the true password he will receive one-time encryptions of junk, but this accomplishes a key reset at the service, rendering the lost OTP's useless. He cannot now use the service until reaching a trusted PC. A user who carries the OTP list both on a cell phone and by paper and who loses the paper can, of course, render the lost sheet useless by using the {`URRSA`}/OTPRefresh service.

5.2 Brute-Force and Denial of Service

An attacker who wishes to guess or brute force the password will gain nothing by going through the service, since we do not protect strong secrets with weak ones. To login normally he would require the userID and password, to login via the service he requires the userID and the password encrypted with the correct key. Any lockout policies enforced by the login server (*e.g.* "Three strikes and you're out") remain in effect. An attacker who observes several logins or gains access to the entire list cannot brute-force the password.

The proxy itself is a likely point of attack. Observe, however, that the OTP Login Web interface (described in Section 4.2) is merely a conventional password web interface: it accepts text and password HTML form fields from the user and relays them to backend processing. Thus the web facing portion of the proxy is implemented with a tried and trusted password server. The fact that we use components with well-understood attack surfaces increases the expectation that attacking the system will be hard. Since at the backend credentials are stored only temporarily, a snapshot of the database would gain an attacker little. A rogue employee at the proxy would see at any given time only the credentials of users currently logging in.

Since we do not authenticate users it is possible to exhaust a user's OTP list by repeatedly invoking the OTPLogon service (with the correct url and userID but incorrect OTP's). This form of denial of service is possible, but gains the attacker nothing unless he can lure the user into typing the true password in the clear.

5.3 Session Hijacking and OTP Stealing

There are two main active attacks on the system: session hijacking and OTP stealing. Session hijacking is not addressed by our technique. Indeed, even long established OTP solutions such as SecurID [3] have this vulnerability. However

session hijacking is a complicated attack that requires code tailored to each target login server. The fact that RSA has had considerable commercial success protecting high-value accounts in spite of this well-known vulnerability suggests that session-hijacking is not a common attack. In addition the techniques suggested in [21], which require explicit out-of-band authorization for important transactions, might present a way of addressing this problem.

OTP stealing is the technique that malware can employ to get OTP's from the user. There are a number of variations; the simplest is to allow a user to connect to {URRSA}/OTPLogin, have him enter the requested OTP and then fail to submit it. If the user assumes that he mis-typed he might try again, and give the attacker a second OTP. Depending on the user the attacker might gain anywhere between one and three OTP's this way (we assume that it is unlikely that a user will type more than that). This is a well-known attack on all static OTP systems; dynamic systems such as securID do not have this vulnerability since each OTP is good for only a few seconds. We do not eliminate this attack. However, we point out that an attacker who logs in with a stolen OTP has full access, but cannot change the account password. This is so, since almost all web services require the original password before allowing a change; while the attacker has one or more OTP's he cannot use these to derive the password. This restricts the attacker: if he has stolen three OTP's he can now login three times. He cannot change or get access to the true password and thus every access must continue to be done via URRSA. It is possible to restrict the types of actions that can be performed via the proxy. For example, submission of the HTML form that changes the user email, address or phone number might be forbidden. This can be accomplished by adding a rule to the translations in Table 2 that drops the request that contains any such POST. Finally, we point out that the OTP stealing attacker leaves a clue to his presence in that he must cause a login to fail for each OTP he steals (*i.e.* each OTP becomes worthless after its first use). Thus, in the absence of failed login attempts the user can be confident that no OTP has been stolen. A user who suspects that an OTP has been stolen can, of course, render them useless by connecting to {URRSA}/OTPLogin.

6 Status and Evaluation

We have implemented the system described and deployed on an internet-facing server. The service currently supports OTP logins to a number of sites. The entire server code is small enough that it can comfortably be hosted on a modest desktop machine. This makes self-hosting a real possibility: *i.e.* users who have a fixed IP address or domain name might run their own instance. This removes the necessity of trusting a proxy maintained by a third party.

The URRSA instance in its current form has been in active use by a small number of users for over six months. A large number of successful logins to have been handled. These were carried out from a variety of networks; *i.e.* machines that exist on home networks, are behind corporate firewalls, internet cafés and public library locations have all been tried. The service works well with Internet Explorer, Firefox, Safari and Opera. Users report that all of the functionality

generally available at a server works well when accessed through the URRSA service. Users do not report perceptible delay, and the service is generally transparent to users once connection is established. The service is running at www.urrsa.com. It is not possible to invite general use as yet. However, for demonstration and verification purposes, we may allow restricted access as conditions permit.

7 Conclusions

We have described a system that allows users one-time password access to accounts without changing the server or the client. The method is entirely general and can be applied to almost any login server. Among the key contributions are a very simple user experience and a truly robust MITM translation. We do not authenticate users: thus there are no additional secrets to remember or tokens for the user to carry. The service acts as a transparent MITM between user and login server: thus there are no browser settings to be done or undone. We employ a simple mapping of the arbitrary input password to restricted character set OTP's: thus every OTP is readable without ambiguity no matter what display or font is used, can be transmitted over SMS, and can be entered even on unfamiliar keyboards without the use of meta keys.

Acknowledgements. the authors wish to thank Eric Lawrence, Ziqing Mao, Nikita Pandey, Erin Renshaw and Dany Rouhana for help with various stages of this work.

References

1. http://labs.zarate.org/passwd/
2. http://www.cl.cam.ac.uk/~mgk25/otpw.html
3. http://www.rsasecurity.com
4. http://www.kyps.net
5. http://www.csoft.co.uk/sms/character_sets/encoding.htm
6. http://www.jmarshall.com/tools/cgiproxy
7. http://www.clickatell.com
8. http://www.fiddlertool.com
9. http://www.xk72.com/charles/
10. http://www.portswigger.net/proxy
11. Herley, C., Florêncio, D.: How To Login From an Internet Café without Worrying about Keyloggers. In: Symp. on Usable Privacy and Security (2006)
12. Cheswick, W.: Johnny Can Obfuscate: Beyond Mother's Maiden Name. In: Proc. Usenix HotSec (2006)
13. Coskun, B., Herley, C.: Can "Something You Know" be Saved? In: Wu, T.-C., Lei, C.-L., Rijmen, V., Lee, D.-T. (eds.) ISC 2008. LNCS, vol. 5222, pp. 421–440. Springer, Heidelberg (2008)
14. Florêncio, D., Herley, C.: A Large-Scale Study of Web Password Habits. In: WWW 2007, Banff (2007)
15. Florêncio, D., Herley, C.: KLASSP: Entering Passwords on a Spyware Infected Machine. In: Jesshope, C., Egan, C. (eds.) ACSAC 2006. LNCS, vol. 4186. Springer, Heidelberg (2006)

16. Florêncio, D., Herley, C., Coskun, B.: Do Strong Web Passwords Accomplish Anything?. In: Proc. Usenix Hot Topics in Security (2007)
17. Gaber, E., Gibbons, P., Matyas, Y., Mayer, A.: How to make personalized web browsing simple, secure and anonymous. In: Proc. Finan. Crypto 1997 (1997)
18. Golle, P., Wagner, D.: Cryptanalysis of a Cognitive Authentication Scheme. In: Symp. on Security and Privacy (2007)
19. Haller, N.: The S/KEY One-Time Password System. In: Proc. ISOC Symposium on Network and Distributed System Security (1994)
20. Herley, C., Florêncio, D.: Phishing as a Tragedy of the Commons. In: NSPW 2008, Lake Tahoe, CA (2008)
21. Jammalamadaka, R.C., van der Horst, T.W., Mehrotra, S., Seamons, K., Venkasubramanian, N.: Delegate: A Proxy based Architecture fort Secure Website Access from an Untrusted Machine. In: Jesshope, C., Egan, C. (eds.) ACSAC 2006. LNCS, vol. 4186. Springer, Heidelberg (2006)
22. Lamport, L.: Password Authentication with Insecure Communication. Communications of the ACM (1981)
23. Luotonen, A.: Web Proxy Servers. Prentice-Hall, Englewood Cliffs (1998)
24. Mannan, M., van Oorschot, P.C.: Using a Personal Device to Strengthen Password Authentication from an Untrusted Computer. In: Dietrich, S., Dhamija, R. (eds.) FC 2007 and USEC 2007. LNCS, vol. 4886. Springer, Heidelberg (2007)
25. Wu, M., Garfinkel, S., Miller, R.: Secure Web Authentication with Mobile Phones. In: DIMACS Workshop on Usable Privacy and Security Software (2004)
26. Mao, Z., Herley, C.: Robust Reverse Proxy Implementation. MSR-TR
27. Pashalidis, A., Mitchell, C.J.: Impostor: A single sign-on system for use from untrusted devices. In: Proceedings of IEEE Globecom (2004)
28. Pering, T., Sundar, M., Light, J., Want, R.: Photographic Authentication through Untrusted Terminals. IEEE Security and Privacy (2003)
29. Schneier, B.: Applied Cryptography, 2nd edn. Wiley, Chichester (1996)
30. Bell, T.C., Cleary, J.G., Witten, I.H.: Text Compression. Prentice-Hall, Englewood Cliffs (1990)
31. Tan, D., Keryana, P., Czerwinski, M.: Spy-resistant keyboard: more secure password entry on public touch screen displays. In: CHISIG 2005 (2005)
32. Weinshall, D.: Cognitive Authentication Schemes Safe Against Spyware. In: Symp. on Security and Privacy (2006)

A Additional Details

A.1 Pseudo-code of the Mapping Procedure

Here is a pseudo code illustrating the above encoding procedure:

```
//Encode
// get bits from input password P
BitString = 0;
TotalBits = 0;
for each character P[i] {
   PP = table128_lookup_character(P[i]);
   BitString = BitString <<7 + PP;
   TotalBits += 7;
}
```

```
//encrypt
Key = get next TotalBits bits from One Time Encryption Pad
BitString = BitString XOR Key;
//convert bits to OTPi characters
i = 0;
while TotalBits > 0 {
     PP = BitString AND 31
     OTPi[i] = table32[PP];
     BitString = BitString >> 5;
     TotalBits -= 5;
     i++;
}
```

A.2 Sequence of Events

1. User navigates to `http://{URRSA}/OTPLogon`, enters the userID and url (*e.g.* `www.paypal.com`). This allows the server to determine the key values, that were used to generate the OTP list.
2. For the k-th login the user enters the k-th one-time password OTPk
3. The server decrypts to get the true password *pwd*
4. Browser is auto-transferred to request the url via the URRSA service (*e.g.* we request `https://{URRSA_1}`).
5. Server requests `https://www.paypal.com`, receives the response and populates login form with userID and "roguePwd," and sends to user.
6. User receives pre-populated `https://www.paypal.com` page and clicks submit button.
7. Server receives request, replaces "roguePwd" with *pwd*, forwards to `https://www.paypal.com`.
8. Login proceeds and server continues in a MITM rôle until the user navigates from the site.

A.3 Login Servers That Do Not Use Forms

Our system is applicable to login servers that use HTML forms and POST a password to be authenticated at the server. While this appears to account for the vast majority of login servers there are exceptions. Certain institutions implement an entirely proprietary authentication on their website using Flash or a comparable technology. For example FirstTech Credit Union uses only Flash on their login page `https://online.firsttechcu.com/`

Table 1. A summary of the services described. The variable `URRSA` represents the host domain name or IP address (*e.g.* we give the hostname of our implementation in Section 6). The last service is requested after an OTP has been received from the user and decrypted.

`http://{URRSA}/OTPRegistration`	Register login domain and userID and receive OTP list
`http://{URRSA}/OTPLogin`	Enter login domain, userID and requested OTP
`http://{URRSA}/OTPRefresh`	Enter login domain, userID and requested OTP to get new OTP list
`https://{URRSA_1}`	MITM service that performs mappings described in Section 4.2 and Table 2

Table 2. The translation rules applied to the request/response stream to reverse proxy between the two hosts `www.paypal.com` and `URRSA_1`

Modify	Search for	Replace with
Request Header	`URRSA_1`	`www.paypal.com`
Response Header	`www.paypal.com`	`URRSA_1`
Response Header	domain=`.paypal.com`	domain=`URRSA_1`
Response Body	`www.paypal.com`	`URRSA_1`
Response Body	`www.paypalobjects.com`	`URRSA_2`
Response Body	type="password"	type="password" value="roguePwd"
Request Header	roguePwd	Actual password as decrypted

Can "Something You Know" Be Saved?

Baris Coskun[1] and Cormac Herley[2]

[1] Polytechnic University, Brooklyn, NY
[2] Microsoft Research, Redmond, WA

Abstract. "Something you know," in the form of passwords, has been the cornerstone of authentication for some time; however the inability to survive replay attack threatens this state of affairs. While "something you know" may always be used in addition to "something you have" we examine whether it can be salvaged as the solo factor for authentication. A recent surge of interest in Challenge Response authentication schemes raises the question whether a secret shared between the user and the server can allow secure access even in the presence of spyware.

Our conclusion is negative. Assuming only a limit on the amount that a user can remember and calculate we find that any scheme likely to be usable is too easily brute forced if the attacker observes several logins. This is true irrespective of the details of the scheme. The vital parameter is the number of bits of the secret involved in each bit of the response. When this number is too low the scheme is easily brute-forced, but making it high makes the scheme unworkable for the user. Our conclusion is that single factor "something you know" schemes have a fundamental weakness unless the number of logins the attacker observes can be restricted.

Keywords: Authentication, passwords, challenge response.

1 Introduction

Authentication is commonly implemented by requiring a user to provide proof of one or more of:

- Something you know (*e.g.* a password)
- Something you have (*e.g.* a smartcard)
- Something you are (*e.g.* a fingerprint).

Passwords have enjoyed a long run as the dominant means of authentication. An active web user today will access financial institutions, social networking, and email accounts using passwords. It is not uncommon for users to have twenty or more password-protected accounts, and to type passwords several times per day. With this success has come a host or problems and attacks. Users notoriously choose weak passwords, potentially opening the door to brute-force attacks. The Phishing phenomenon has shown that users can be lured into divulging their passwords to sites masquerading as the real login server. Malicious spyware such

T.-C. Wu et al. (Eds.): ISC 2008, LNCS 5222, pp. 421–440, 2008.

as a keylogger can capture the password and thereby afterward allow an attacker access to the protected account.

Despite these problems passwords are still almost universally used for account access, even when assets of considerable value are protected. A majority of the large banks and financial institutions in the US appear to use password only access. The reasons for the success are clear: passwords are a simple and well-understood technique. They allow institutions to offer users 24/7 access to their accounts from any browser. No special hardware or training is required, and they rely on memory only. In addition it is argued in [9] that losses from password stealing attacks may not be as high as often assumed.

Our focus in this paper is on the failure of passwords to resist replay attack. That is, a single login from a spyware infected machine gives an attacker everything he needs to gain access to an account. If an unsuspecting user accesses their bank account from an infected internet cafe machine it is easy for the spyware to gather the userID and password and then access the account when the attacker chooses.

There has been a great deal of password related work recently, which we attempt to review in Section 2. Our goal in this paper is to evaluate mechanisms for access control from untrusted machines that rely only on the memory and calculating power of the user. Examples are the recently proposed scheme of Weinshall [17], the Virtual Password work of Lei *et al.* [12], the picture authentication work of Pering *et al.* [15], and the work of Cheswick [3]. Each of these approaches attempts to allow users to login securely from untrusted machines. That is, spyware running on the machine and observing a login should not be able to use that information to gain access to the account. To be useful, the scheme should allow several logins from the same machine without giving the attacker enough information to gain access. Golle and Wagner [7] for example recently showed that the Weinshall scheme could be broken after observing 7 or so logins. We show how to break the Virtual Passwords scheme of [12] in Section A.

The constraints that we impose are that the scheme should rely only on the user's memory and calculating ability. As we will see in Section 2 none of the solutions proposed so far to the untrusted login problem possess all of the desired features mentioned above. Our goal is to determine whether it is possible to construct a scheme which satisfies all of the requirements. For example, can the break in Weinshall's scheme be fixed or is it a fundamental hole? The pattern of progress on this question has been of suggested solutions, such as [17], followed by rebuttals, such as [7]. This is an unsatisfying state of affairs. The problem is important enough that researchers often return to it in hopes of a solution. Yet, in breaking such a scheme we seldom find out whether that particular scheme was flawed or whether there is a fundamental limitation that prevents us from designing a "Something You Know" scheme that will withstand determined attack. This paper demonstrates that the problem is truly a fundamental one: Weinshall [17] and the other schemes commit the common error of failing to involve enough bits of the secret in calculating each bit of the output. For any scheme that fails to do so a brute-force attack can determine the secret given

enough observed logins. The only ways to avoid this is to involve more bits of the secret per output bit, increase the size of the secret, or restrict the number of logins observed. But we show that increasing either the secret size or number of bits used places an insupportable burden on the user: for example to involve each bit of a 60-bit secret in each bit of a 20-bit login would require the user to make two binary decisions per second at a sustained rate for 10 minutes in order to login.

2 Related Work

2.1 Challenge Response Authentication

A scheme introduced by Weinshall [17] assigns a user thirty images to memorize. When logging in the user is presented with a palette containing 8×10 images. Starting at the top left corner he calculates a path by moving down or to the right depending on whether the current image is one of his thirty images. On reaching the bottom or right edge of the palette the path exits and he enters a 2-bit number to indicate the exit point. This is performed 11 times for a 22-bit login. The scheme was broken using SatSolver by Golle and Wagner [7].

Pering *et al.* [15] suggest having the user upload a number of his own images. When logging in he is presented with a palette of four images, one of which is his. He selects his image (thus effectively giving away 2 bits) and repeats 10 times for a 20 bit login. Clearly the user must upload to the server at least $10\times$ as many images as the attacker will observe logins.

Lei *et al.* [12] propose a scheme to secure users' passwords in spyware infected environments. The user must calculate response symbols that are derived from symbols of a fixed password and a challenge symbol string using modular arithmetic. The complexity of the calculation appears considerable. The authors suggest a helper application can be used to assist the user in performing the calculations. We show how this scheme can be broken in Section A. Cheswick [3] explores the general feasibility of allowing multiple logins from a single shared secret. He proposes a number of approaches, but draws no definite conclusion on whether the goal is attainable or not.

2.2 Logging in from Untrusted Machines

Pashalidis and Mitchell [14] propose a single sign-on system as a means of evading spyware on an untrusted machine. Credentials are stored in the cloud and a challenge response authentication is used to grant access. The user is assigned a secret string, and when accessing the credentials is challenged to produce the characters at three randomly chosen positions in the string.

Florêncio and Herley [5] propose a proxy-based solution where the user maps the password in a fashion that is unmapped by a MITM proxy before forwarding to the login server. The same authors suggest [2] a simple trick that allows a user to evade current generation keyloggers when entering a password on an untrusted machine.

Lim [13] proposes a scheme to prevent spyware from capturing screenshots. The user enters data from an on-screen keyboard. Rather than comprising a single image the on-screen keyboard is formed from several images displayed in rapid succession. A single screen shot at the time of the mouse click will not tell the spyware what key was selected, while capturing many images will force the spyware to consume resources thereby risking discovery.

One-time password systems such as S/Key [11,8] resist replay. SecurID [1] is a well known commercial product that generates time-evolving one-time codes on a keychain device that match codes generated at the server. In an enhancement of [5] Florêncio and Herley [4] propose a proxy-based solution where the user enters an encrypted version of the true password that is decrypted by the proxy before forwarding to the login server. As with any one-time password system the user must carry either a list of the one-time passwords, or a device that calculates them. The scheme of [4] differs from [11,8,1] in that it works with existing password servers without modification.

2.3 Alternatives to Passwords

Passwords are so widely used, and attacks on them so varied that a large literature has grown around addressing these attacks. We can give only a small sample of recent password related work. Graphical passwords [10,16] address the oft-cited problem with passwords that users have too many passwords to remember and often make weak easily guessed choices. This approach promises to improve the password strength and memorability, it of course does nothing to address the replay attack. Florencio *et al.* [6] argue that passwords strength is of limited importance when password stealing techniques such as phishing and keylogging are the main threats.

Two-factor authentication is the practice of requiring possession of a piece of hardware, or a biometric before allowing login. While far more secure tokens often require an issuing authority, and require that the user carry something. Our work can be seen as an examination of whether one factor schemes can ever truly resist replay.

3 Challenge-Response Schemes

Shared secret (*i.e.* "Something you know") schemes require that a user prove that she knows the secret before access is granted to an account. Passwords are the simplest case, since entry of the password, $\mathbf{P}$, causes the server to conclude that the request for access has come from the correct user. However, since the secret is not dynamic, a single observation suffices to allow an attacker to break the system. One-time password systems, by contrast, deny an attacker useful information, even assuming he observes a login. Here, the user possesses a list, rather than a single password, and enters the i-th password, $\mathbf{P}_i$, on demand from the server. The successive passwords $\mathbf{P}_i$, can be derived from a single secret, as with S/Key [11,8], dynamically generated on-the-fly as with SecureID

[1], encrypted versions of a single static password as in [4] or just a pre-computed set of random strings. In each case, of course, the user must carry the list of one-time passwords (or the device that calculates them). We consider it unreasonable to expect that a user will commit to memory a series of passwords each of which will be used only once.

With a challenge response scheme the user again shares a secret $\mathbf{S}$ with the server, but to login demonstrates that she knows the secret without revealing the secret itself. Instead of delivering the entire secret, the user delivers something derived from the secret in response to a challenge from the server. Thus the server produces a random challenge $\mathbf{C}_i$ and requests that the user return $f(\mathbf{S}, \mathbf{C}_i)$, where $f()$ represents a calculation that the user must perform (of which more below). The server will produce a new challenge for each login, and hence the attacker cannot simply replay an observed response, since

$$f(\mathbf{S}, \mathbf{C}_i) \neq f(\mathbf{S}, \mathbf{C}_j).$$

The basic outline of the three authentication schemes discussed is given in Figure 6.

While replay may not be possible on a challenge response scheme there is still the possibility that given several observations $f(\mathbf{S}, \mathbf{C}_i)$, for $i = 0, 1, 2 \cdots$ the attacker may be able to determine the secret $\mathbf{S}$. Clearly this depends on the function $f()$. Ideally, an attacker should be unable to determine the secret even after observing many logins. If such a scheme exists it could be enormously beneficial. It would share with passwords the fact that it is memory based and requires no hardware. And yet it would entirely solve phishing, keylogging and other password stealing attacks.

3.1 General Setup and Notation

We will assume that during a registration process the user and server agree to share an N-bit secret $\mathbf{S}$. For the i-th login the user provides the userID and receives a randomly chosen challenge $\mathbf{C}_i$ from the server. Given this challenge, she must calculate, and deliver to the server, the M-bit response

$$\mathbf{R} = f(\mathbf{S}, \mathbf{C}_i).$$

This model is sufficiently general to cover the major existing challenge response proposals *e.g.* [17,3,15,12].

The Secret. The size, N, of the secret space must be large. Recall that $\mathbf{C}_i$ is known to the attacker, and of course $f()$ must be public. If N is small enough the attacker might just list the response

$$\mathbf{R}' = f(\mathbf{S}', \mathbf{C}_i)$$

for each of the 2^N possible secrets. Any $\mathbf{S}'$ for which $\mathbf{R}' = \mathbf{R}$ is a candidate for the true secret. There would be only 2^{N-M} candidate secrets after observing a

single response. Subsequent logins would narrow the field further. The key to avoiding this attack is that 2^N must be too large for an attacker to list. The tradeoff, of course, is that N must be small enough for the user to remember and perform calculations on. A 256-bit secret space will resist enumeration, but this is equivalent to a 77-digit PIN, and is probably far too much for a user to remember. How many bits a user can be expected to remember and perform calculations on is somewhat representation dependent. In theory users might be able to remember an $N = 80$ bit secret (*i.e.* equivalent to an 24-digit PIN), but it is hard to argue that they could also reliably perform calculations on such a large secret. Weinshall [17] makes both the memorization and calculation tasks simpler by making the user memorize the secret in a set membership format. At the lower end if a machine can evaluate the response to 2^{10} challenges per second (this is approximately the rate we achieved with an implementation of the Weinshall scheme [17]) then even a secret space of size $N = 37$ can be exhaustively searched by a single machine in a single year. Since enumerating responses to a challenge is a task easily divided among many machines we probably have to consider secrets on the order of 50 bits as the minimum that can even be considered for successive logins on a compromised machine. Thus we get a probable range for the secret space, bounded below by 50 bits as the minimum to resist enumeration and above by 80 bits as probably the most that a human can be expected to remember and calculate on.

The Response. Of course $\mathbf{R}$ should be drawn from a large enough space to make random guessing unlikely to succeed. For example, if $M = 20$ then a single random guess at $\mathbf{R}$ has only a one in a million chance of succeeding. Since the response must be entered using keyboard and/or mouse $\mathbf{R}$ is generally delivered as a series of T symbols $\mathbf{R} = R(0)R(1)R(2)\cdots R(T-1)$, where each $R(t)$ is a k-bit symbol and $kT \approx M$. For example, the response to the challenge might be a 6-digit number since $\log_2(10) \cdot 6 \approx 20$. In [17] the response is a series of $T = 11$ 2-bit symbols, giving a 22-bit effective login.

The Challenge. The challenge is the random component that makes each response different. The form that it takes depends on the form in which the secret is stored. The main requirement is that the server have a large enough suite of challenges. If there are fewer than 2^M challenges then we have unnecessarily reduced the size of the output response space (making the attackers guessing task easier).

The Calculation. Without loss of generality we'll say that each response symbol is produced by a separate calculation function: $R(t) = f_t(\mathbf{S}, \mathbf{C}_i)$. Thus overall

$$\mathbf{R} = [f_0(\mathbf{S}, \mathbf{C}_i) f_1(\mathbf{S}, \mathbf{C}_i) f_2(\mathbf{S}, \mathbf{C}_i) \cdots f_{T-1}(\mathbf{S}, \mathbf{C}_i)].$$

Each of the $f_t()$ represents a calculation the user must perform, using the secret and the challenge, to produce the response symbol $R(t)$. In general it will involve fewer than N bits of the secret. This is so, since if each bit of $\mathbf{R}$ depends uniformly at random on each bit of $\mathbf{S}$ the user must carry out at least $M \cdot (N-1)$ binary

decisions (see Section 4.4). For an 80-bit secret and a 20-bit response, this would be 1600 binary decisions for a single login. Even if a user could reliably make 2 decisions per second the login would take 13.3 minutes, which is absurd. Thus, we will assume, in general, that $U \leq N$ of the bits of $\mathbf{S}$ are used in the calculation $f_t()$ of each of the response symbols $R(t)$. The lower U the easier the task for the user. It is clear that $U = N$ represents an insupportable burden. However, as we will see in Section 4, making U too small invites a divide-and-conquer attack. U will play a vital role in the tradeoff between usability and security.

We emphasize that the calculation $f(\mathbf{R}, \mathbf{C}_i)$ must be simple enough for a user to perform quickly and accurately. The user must not employ any calculating devices hosted on the untrusted machine. For example, using even a software calculator on the untrusted machine (such as the scheme of [12] suggests) would be unacceptable, since the attacker would have access to the input as well as the output.

We also rule out using the assistance of a calculator on a cell phone, mp3 player or other device, but for a different reason. If the user has access to a cell phone it makes more sense to carry a list of one-time passwords than to perform a challenge response calculation. Our goal is to determine whether a memory alone challenge response scheme can resist attack.

Example Schemes. The described notation and framework are applicable to all challenge-response authentication schemes although the mapping function is different in each scheme. For instance in Weinshall's scheme [17] briefly described in Section 2, the shared secret is the set of 30 images selected among 80. Here the size of the secret is $N = \log_2 \binom{80}{30} \approx 73$ bits. C_i is the permutation of the 80 images which is currently being displayed on the panel and the mapping function M is the mental path that the user is supposed to follow. The response $R(t)$ is the label through which the path drawn by the user exits the panel. Since $R(t)$ can be either 0,1,2 or 3, it is a $k = 2$ bit response. Finally, the server requires 11 challenge-response rounds, which makes $T = 11$.

3.2 Two Trivial Solutions

Hand Over the Bits Unmodified. A trivial solution is to have $f_t()$ just select k bits of the secret. For example, the first time the user logs in she might be asked to enter the first M bits of the secret $\mathbf{S}$, the second time the next M and so on. This has the merit of being simple, but this is good for only N/M logins on the same machine. After that the attacker knows the entire secret and those bits cannot be re-used. In fact this is equivalent to a one-time password system. A variant of this is used in [15], since the user gives away bits each time he indicates an image.

Challenge for Random Bits of Secret. A related alternative is to prompt the user for specific randomly chosen portions of the secret, which are delivered unmodified (this is suggested *e.g.* in [14]). For example, if the secret were remembered in the form of a 24-digit number the server might challenge for 6

randomly chosen digits of the secret (thereby giving a 20-bit login). This has the merit of again being simple, but extending the number of logins that can be achieved. However the probability that the attacker who has observed n logins possesses any particular digit is

$$Pr\{\text{Attacker knows digit}|W \text{ logins}\} = 1 - (1 - M/N)^W.$$

For example, after 8 logins he knows each digit with 90% probability. At this point he has a 53% probability of being able to answer any given challenge successfully. If he gets three attempts before the account is locked he has a greater than 90% chance of being able to successfully respond to one of the challenges.

What's Wrong with These Solutions. It is apparent that in order to withstand more than $\frac{N}{M}$ challenge-response rounds, the server should never ask for the secret bits themselves. For this purpose a carefully designed $f()$ function, which proves to the server that the user knows the secret without revealing the secret bits, is required. Such an $f()$ function should be one-way in the first place so that it should be very hard to determine the input given the output of the function. Otherwise, the scheme would be equivalent to the case where the user submits secret bits as the response.

3.3 Attack Model

First we introduce our attack model. We assume that the attacker has installed spyware on the untrusted machine and observes everything that happens there. All keystrokes, all mouse-movements, everything that goes on the screen, and all network traffic is available to the attacker.

Thus, following the general pattern of a challenge response scheme in Figure 6 the attacker will observe both the challenge from the server $\mathbf{C}_i$ and the client's response $\mathbf{R} = f(\mathbf{S}, \mathbf{C}_i)$. We further assume that the calculating function $f()$ is public, so that the secret $\mathbf{S}$ is the only thing concealed from the attacker. Since we desire that the user be able to login multiple times we will assume that the attacker observes a series of W logins and gets the MW-bit stream:

$$\mathbf{\Gamma} = \mathbf{R}_0\mathbf{R}_1\mathbf{R}_2\cdots\mathbf{R}_{W-1}.$$

Since he has seen the response the attacker can try as many off-line guesses at the secret as computation will allow. That is he can calculate

$$\mathbf{\Gamma}' = \mathbf{R}_0'\mathbf{R}_1'\mathbf{R}_2'\cdots\mathbf{R}_{W-1}',$$

for as many trial secrets $\mathbf{S}'$ as he wishes. If he finds an $\mathbf{S}'$ for which $\mathbf{\Gamma}' = \mathbf{\Gamma}$ he has not necessarily found $\mathbf{S}$. For example, after a single login (*i.e.* $W = 1$) there would be 2^{N-M} secrets that produce the same response as $\mathbf{S}$. However as the number of observed logins increases collisions decrease rapidly. When $MW > N$, *i.e.* the total number of observed bits is greater than the secret size, collisions are negligible and we do not consider them further.

We Cannot Conceal How Many Bits are Used. Recall, from Section 3.1 that the number of bits of the secret U involved in each output bit is an important parameter. This cannot be concealed from the attacker. We can measure the effective number of bits used by a given scheme as follows. Choose two secrets $\mathbf{S}$ and $\mathbf{S}^{'}$ which differ by one bit. Generate A randomly chosen challenges and count the number, B, for which the two secrets produce different responses. Thus the measured fraction of responses where the two secrets produce the same response is $(A - B)/A$. We have already seen (in the case of k-bit symbols) that the expected value is $1 - U/N \cdot (1 - 1/2^k)$. Thus if we compare the expected and actual values we can estimate U as

$$U = \frac{NB}{A} \cdot \frac{1}{1 - 1/2^k}. \tag{1}$$

Of course A and B should be large enough to generate a stable estimate of the actual fraction of responses that remain unchanged. We can use this to estimate the effective number of bits used, for example in the Weinshall [17] scheme, where we find $U_{eff} \approx 7.8$. We will see that this is the fatal weakness, not just of this scheme, but of any scheme which does not make absurd calculating requirements of the user.

4 A Brute Force Attack

We now show how secret can be identified within a reasonable time when the number of bits, U, used to calculate an output symbol is small. The weakness is that when $U \ll N$ similar secrets produce similar responses. The outline of the attack is as follows:

- When secrets are close the responses are close
- It's easy to find a secret that's close
- Once close, it's easy to get closer.

Taken together this gives that the only escape from brute-force attack is to make U large. But we show in Section 4.4 that this forces the burden of calculation on the user to be infeasible.

4.1 When Secrets Are Close the Responses Are Close

First suppose that $\mathbf{S}$ is the user's secret and $\mathbf{S}^{'}$ is an unrelated secret chosen at random. We expect that the probability that two symbols from their respective responses to $\mathbf{C}_i$ are the same is $Pr\{R(t) = R(t)^{'}\} = 1/2^k$. This merely says that all response symbols are equally likely and, for unrelated secrets, the response symbols coincide at random. However, when $\mathbf{S}$ and $\mathbf{S}'$ are close and $U \ll N$ this is no longer true: suppose they differ by e bits *i.e.* $|\mathbf{S} - \mathbf{S}^{'}| = e$. Now, since only a U-bit portion of the N-bit secret is used to calculate each response symbol this portion might include none of the different bits. For example, if $\mathbf{S}$ and $\mathbf{S}^{'}$

differed in only a single position (*i.e.* $e = 1$) we would have $R(t) = R^{'}(t)$ unless that single bit was one of the U bits used in this calculation.

In general the probability that none of the e bits where $\mathbf{S}$ and $\mathbf{S}'$ differ are included in any of the U bits used to calculate $R(t)$ is:

$$\frac{N-e}{N}\frac{N-1-e}{N-1}\cdots\frac{N-(U-1)-e}{N-(U-1)} = \left(\prod_{j=0}^{U-1}\frac{N-j-e}{N-j}\right).$$

When this occurs $R(t) = R^{'}(t)$. Otherwise the responses might still be the same with probability $\frac{1}{2^k}$. Hence the probability of having $R(t) = R^{'}(t)$ given that $|\mathbf{S} - \mathbf{S}'| = e$, can be written as:

$$p_e = \left(\prod_{j=0}^{U-1}\frac{N-j-e}{N-j}\right) + \left(1 - \left(\prod_{j=0}^{U-1}\frac{N-j-e}{N-j}\right)\right)/2^k\,. \tag{2}$$

We graph this in Figure 1. As e decreases (*i.e.* $\mathbf{S}$ and $\mathbf{S}^{'}$ get closer) p_e deviates from $\frac{1}{2^k}$ and gets closer to 1 as shown. When e is very small the probability that $R(t) = R^{'}(t)$ is almost 1. This is important since it says that the probability that the response symbols are equal increases as the distance between the secrets decreases.

Now let's examine the consequences for the attacker who gathers TW response symbols from the W observed login events. Define $\text{Simm}(\mathbf{\Gamma}, \mathbf{\Gamma}^{'})$ as the number of symbol positions in which the two response streams agree; this ranges between 0 and TW. Now, $Pr\{\text{Simm}(\mathbf{\Gamma}, \mathbf{\Gamma}^{'}) = d\}$ is binomially distributed with probability p_e given in (2):

$$Pr\{\text{Simm}(\mathbf{\Gamma}, \mathbf{\Gamma}^{'}) = d \mid |\mathbf{S} - \mathbf{S}^{'}| = e\} = B_{pdf}(d, TW, p_e).$$

In Figure 2 (a) we show how this is influenced by distance (for $N = 80, U = 10, k = 2$ and $W = 10$ logins) by showing the distributions for $e = 12$ and $e = 40$. Observe that the mean of $\text{Simm}(\mathbf{\Gamma}, \mathbf{\Gamma}^{'})$ for secrets that are distance 40 from $\mathbf{S}$ is $TWp_{40} = 100 \times 0.25 = 25$ while for secrets at distance 12 it is $TWp_{12} = 100 \times 0.413 = 41.3$. Also observe that for a given scheme (*i.e.* N, U and k fixed) the separation increases with the number of logins observed. Figure 2 (b) we show the same scheme as in (a), but now with $W = 20$ logins. Clearly it gets easier to tell secrets that are close to $\mathbf{S}$ from those that are far when the attacker observes more logins. Thus the attacker is always assisted by more observations: the more logins he observes the greater the separation between the binomials and the more reliably he can distinguish secrets that are close from secrets that are far from the true secret.

We emphasize that when $\text{Simm}(\mathbf{\Gamma}, \mathbf{\Gamma}^{'})$ is large it does not imply that $|\mathbf{S} - \mathbf{S}^{'}|$ is small, but it does mean that the probability is higher. Thus, among a large enough collection of secrets it is possible to distinguish *in probability* those that are close to the true secret from those that are far.

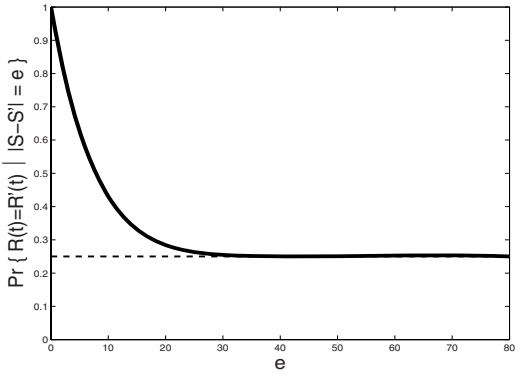


Fig. 1. Probability that an output symbol $R(t)$ is the same for two N-bit secrets that differ in e positions: $Pr\{R(t) = R(t)^{'} \mid |\mathbf{S} - \mathbf{S}^{'}| = e\}$ vs. e. We are using $N = 80$, $U = 10$ and $k = 2$. Observe that when secrets are close (*i.e.* e small) the probability of the outputs being equal is high.

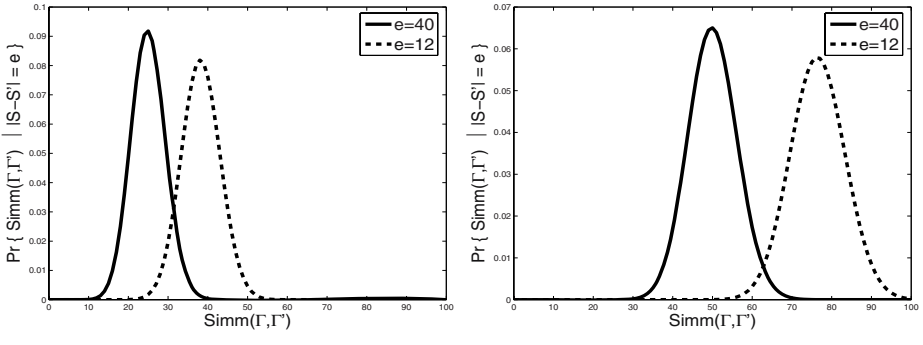


Fig. 2. Secrets that are close produce responses that are close. The expected number of positions where the responses are the same for depends on distance from the true secret. Using $N = 80$, $U = 10$ and $k = 2$ and secrets that are 40 and 12 bits from the true secret. (a) TW = 100 (*i.e.* 10 logins each with 10 2-bit symbols). (a) TW = 200 (*i.e.* 20 logins each with 10 2-bit symbols).

4.2 It's Easy to Find a Secret That's Close

If the attacker has a large collection of candidate secrets he now has a good strategy to tell secrets that are close from secrets that are far. By selecting only those points for which

$$\text{Simm}(\mathbf{\Gamma}, \mathbf{\Gamma}^{'}) \geq \text{Threshold} \tag{3}$$

we can remove from consideration those that are likely to be far away. For example, in Figure 2, if we threshold at TWp_{12}, *i.e.* the mean of the p_{12} binomial, we include a fraction $1 - B_{cdf}(TWp_{12}, TW, p_{12}) = 0.5$ of secrets that are distance 12 from $\mathbf{S}$ and only $1 - B_{cdf}(TWP_{12}, TW, p_{40}) = 0.0065$ of those that are distance

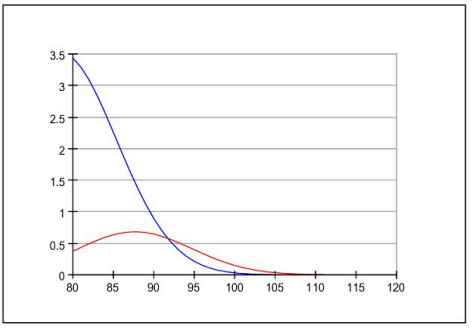


Fig. 3. Figuring out which of your neighbors take you closer to/away from the true secret. The graph shows that even the binomial pdfs with probabilities p_{e+1} and p_{e-1} are separated when e is small enough. The scheme shown is for $N = 73, U = 8.8, k = 2$ with 10 logins.

40. Thus, at a threshold of TWp_{12} the vast majority of distance-40 points are discarded, while only 0.50 of distance-12 points are. Of course there are many more secrets at distance 40 than at distance 12, *i.e.* $\binom{N}{40} \gg \binom{N}{12}$. But if the attacker starts with a large enough collection of candidate secrets $\mathbf{S}'$ he can quickly trim the collection to include only those that have a higher probability of being close. Of the 2^N total secrets in the space the number of points that satisfy (3) (*i.e.* have response streams that are close to that produced by the true secret) is given by summing the tails of the binomials multiplied by the number of points:

$$\sum_{k=0}^{N} \binom{N}{k} \cdot (1 - B_{cdf}(\text{Threshold}, TW, p_k)).$$

A total of $\binom{N}{e}$ secrets will live within an e-ball of the user's secret. Thus we should expect that if the attacker selected

$$2^N \Big/ \binom{N}{e}$$

secrets at random, at least one would be a distance e from the user's secret $\mathbf{S}$. Of these we expect a fraction

$$\frac{\sum_{k=0}^{N} \binom{N}{k} \cdot (1 - B_{cdf}(\text{Threshold}, TW, p_k))}{2^N}$$

to satisfy (3).

For simplicity, let's choose Threshold $= TWp_e$ so the attacker retains 50 % of the points that are a distance e from the true secret. Thus for a cost of $2^N/\binom{N}{e}$ response evaluations the attacker will end up with

$$\left(\sum_{k=0}^{N} \binom{N}{k} \cdot (1 - B_{cdf}(TWp_e, TW, p_k)) \right) \Big/ \binom{N}{e}$$

secrets with a 50 % probability that one of them is a distance e from the true secret. Every doubling the number of points examined reduces the probability that he misses the true secret by a factor of 2. Taking $2^{N+Q}/\binom{N}{e}$ evaluations improves the chances that includes at least one point within distance e of the true secret to $1-(1-0.5)^Q$.

4.3 Once Close, It's Easy to Get Closer

Suppose we have $\mathbf{S}'$ such that $|\mathbf{S}-\mathbf{S}'|=e$, where e is small. How do we now find $\mathbf{S}$? First consider the N secrets that are distance 1 from $\mathbf{S}'$, *i.e.* consider $\mathbf{S}''$ such that $|\mathbf{S}'-\mathbf{S}''|=1$. For e of these we will have

$$|\mathbf{S}-\mathbf{S}''|=e-1,$$

and for the remaining $N-e$ we have

$$|\mathbf{S}-\mathbf{S}''|=e+1.$$

That is, each distance-1 neighbor of $\mathbf{S}'$ is either a distance-$(e-1)$ or a distance-$(e+1)$ neighbor of the true secret $\mathbf{S}$.

Of course the attacker doesn't know which $N-e$ of the N neighbors are closer, and which e are further away. But he does know that the responses $\mathbf{\Gamma}'$ that are closest to $\mathbf{\Gamma}$ are more likely to come from the distance-$(e-1)$ neighbors. This is made explicit in Figure 3 where we plot

$$Pr\{\mathrm{Simm}(\mathbf{\Gamma},\mathbf{\Gamma}')=d \mid |\mathbf{S}-\mathbf{S}''|=e+1\}=B_{pdf}(d,TW,p_{e+1})$$

and

$$Pr\{\mathrm{Simm}(\mathbf{\Gamma},\mathbf{\Gamma}')=d \mid |\mathbf{S}-\mathbf{S}''|=e-1\}=B_{pdf}(d,TW,p_{e-1})$$

for the scheme as in Figure 2 (*i.e.* $N=80, U=10$ and $k=2$) and $e=12$. Thus, while the tails of the binomials overlap, it is more likely, for example, that if $\mathrm{Simm}(\mathbf{\Gamma},\mathbf{\Gamma}')\geq 95$ that $\mathbf{S}'$ is a distance-$(e-1)$ neighbor of $\mathbf{S}$.

Suppose $\mathbf{S}'$ is a distance-e neighbor of $\mathbf{S}$. Now, if the attacker retains (among the N distance-1 neighbors of $\mathbf{S}'$) the three that maximize $\mathrm{Simm}(\mathbf{\Gamma},\mathbf{\Gamma}'')$ it is overwhelmingly probable that he retains at least one distance-$(e-1)$ neighbor of $\mathbf{S}$. We quantify this in Figure 4 which shows the probability of one of the e secrets that are closer to $\mathbf{S}$ being among the three that maximize $\mathrm{Simm}(\mathbf{\Gamma},\mathbf{\Gamma}'')$. As can be seen for small enough e the attacker is almost guaranteed to have at least one point that is closer among the top three. For example, using the $W=20$ login plot of Figure 4 if $|\mathbf{S}-\mathbf{S}'|=17$ and we choose the three distance-1 neighbors of $\mathbf{S}'$ that have responses closest to $\mathbf{\Gamma}$ the probability is 0.997 that one of them is closer to the true secret (*i.e.* for one of them $\mathbf{S}''$ $|\mathbf{S}-\mathbf{S}''|=16$). Thus given a secret that is close, the attacker has a high probability of getting one point (among three) that is closer still. Since, this probability increases as e decreases, the closer he gets the easier it gets. Now the attacker need merely iterate.

At worst, this would give the attacker 3^e points to consider before finding the secret, which is of course a huge improvement over $\binom{N}{e}$. In fact we can do better; rather than retain 3^m points for each $m = 0, 1, \cdots e$ we retain only the 10 points that maximize $\text{Simm}(\mathbf{\Gamma}, \mathbf{\Gamma}^{''})$. Figure 5 shows the probability that this simplification finds the correct secret as a function of e. To calculate this curve we measured the number of times that, starting at $\mathbf{S}^{'}$, this algorithm found it's way to $\mathbf{S}$ given that $|\mathbf{S} - \mathbf{S}^{'}| = e$. This was done numerically, by generating random N-bit secrets, and challenges that use a randomly chosen U bits of the secret. The k-bit symbols were generated uniformly at random from the U challenge bits, and we repeat TW times to simulate a stream of W logins. As can be seen the attacker can find his way "home" by, starting at $\mathbf{S}^{'}$, retaining the 10 secrets that maximize $\text{Simm}(\mathbf{\Gamma}, \mathbf{\Gamma}^{''})$.

4.4 Putting the Pieces Together

We put the pieces of the above analysis together to form a generic brute-force attack to reveal the user's secret $\mathbf{S}$ given the observation from W logins $\mathbf{\Gamma}$. Essentially the attacker chooses enough brute force points to ensure that several of them are close and applies the test of (3) to retain only those that are probably close. On all of the retained points he attempts to iterate and get closer. Those points that actually are close will converge to the true secret. This leads to the following algorithm.

- foreach($2^{N+Q}/\binom{N}{e}$ random secrets $\mathbf{S}^{'}$){
- for ($m = 0, 1, \cdots, e-1$){
- Calculate $\text{Simm}(\mathbf{\Gamma}\mathbf{\Gamma}^{''})$ of the distance-1 neighbors of each element in list
- Retain the 10 secrets that maximize $\text{Simm}(\mathbf{\Gamma}, \mathbf{\Gamma}^{''})$.
- if ($\text{Simm}(\mathbf{\Gamma}, \mathbf{\Gamma}^{''}) = TW$ for any list secret) break }}

Since we keep 10 secrets in the list, and each secret has N distance-1 neighbors this algorithm requires $10Ne$ evaluations.

Our fundamental unit of complexity is an evaluation of $\mathbf{\Gamma}^{'}$ for a given secret $\mathbf{S}^{'}$. The overall complexity is the cost of the brute-force search plus the cost of finding the true secret from the points that survive the threshold test:

$$\left(2^N + 10 \cdot N \cdot e \cdot \sum_{k=0}^{N} \binom{N}{k} \cdot (1 - B_{cdf}(TWp_e, TW, p_k))\right) \Big/ \binom{N}{e}. \quad (4)$$

The complexity is inversely related to e. The brute-force (left-hand) term dominates for small e. For example, if $e = N/2$ even a small collection of randomly chosen secrets will contain one e-neighbor, while if $e = 1$ we must include almost the whole secret space.

Choosing the largest possible e for a given scheme will minimize the attacker's complexity. Of course, the attacker must limit his choice of e to those that allow reliable zooming in on the true secret once a distance-e neighbor has been found. For example, in the scheme shown in Figure 5 at $e = 17$ the attacker is almost

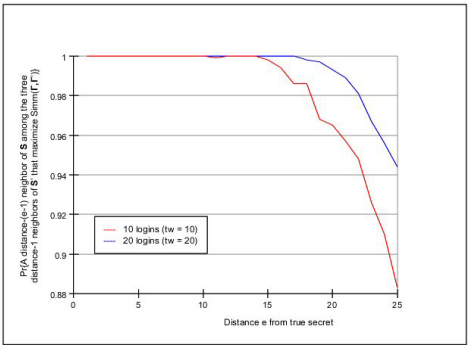


Fig. 4. Given a secret $\mathbf{S}'$ that is distance e from the true secret $\mathbf{S}$ how easy is it to get closer still? The graph shows the probability that the attacker finds a secret distance $e-1$ from $\mathbf{S}$ if he chooses the three distance-1 neighbors of $\mathbf{S}'$ that produce responses most like the observed response. The scheme shown is for $N = 73, U = 8.8, k = 2$ with 10 and 20 logins (left and right resp).

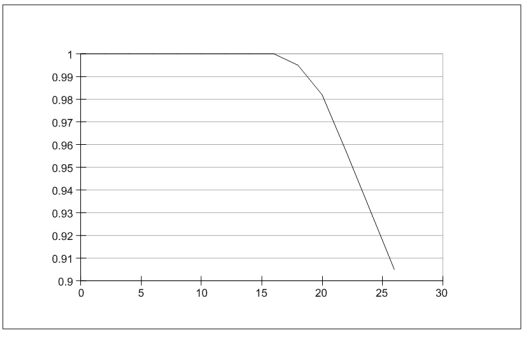


Fig. 5. Given a secret $\mathbf{S}'$ that is distance e from the true secret $\mathbf{S}$ how easy is it to get closer still? The graph shows the probability that by retaining the 10 points that maximize Simm($\mathbf{\Gamma}, \mathbf{\Gamma}''$) and iterating that we find our way from $\mathbf{S}'$ to the true secret. This reduces the complexity from 3^e to $10 \cdot N \cdot e$ points that must be searched. The scheme shown is for $N = 73, U = 8.8, k = 2$ with 10 logins.

guaranteed to successively find secrets that are $16, 15, \cdots 2, 1$ from the true secret, while at $e = 35$ this is extremely unlikely to happen.

We evaluate numerically the largest e that gives the algorithm of Section 4.3 a 0.975 probability of converging to the true secret from a distance e. The results are tabulated in Table 1. As can be seen when $U = 5$ the attacker can start quite far away (*i.e.* e large) and still find the secret, while for $U = 10$ he must start a great deal closer. The table also makes clear that the more logins the attackers observes the easier his task gets (*i.e.* as W increases so also does the value of e from which he can reliably expect to find the secret).

Table 1. The largest value of distance e to have the algorithm of Section 4.3 find the true secret with probability 0.975. We evaluate this for $U = 5$ (left) and $U = 10$ (right). Clearly, the more logins W the attacker observes the easier fidning the true secret becomes. Also, for small U almost any starting point will allows convergence to $\mathbf{S}^{'}$. But even at $U = 4$ a human will require at least 60 seconds to calculate the response to a challenge.

W	$N = 50$	$N = 60$	$N = 70$	$N = 80$
10	19	21	26	39
15	20	29	33	39
20	23	29	34	39
25	24	29	34	39

W	$N = 50$	$N = 60$	$N = 70$	$N = 80$
10	10	11	15	16
15	11	14	16	17
20	13	15	17	18
25	14	16	18	20

Complexity of the User's Task. In Section 4.1 we showed that secrets that are close produce responses that are close. We used the fact that if none of the e bits where $\mathbf{S}$ and $\mathbf{S}^{'}$ differ is among the U bits used to calculate a particular output symbol then the two secrets produce the same output symbol $R(t)$. However, if any of the e bits where they differ was involved we assumed that all outputs were equally likely. So the output would be the same with probability $1/2^k$. Thus, for secrets that differ from $\mathbf{S}$ in any of the U bits used to calculate $R(t)$ the output symbol $R^{'}(t)$ has a uniform random distribution. This implies that each of the k bits of the output symbol depends on all U input bits, but is independent of the other $k-1$ output bits. Hence at least $k(U-1)$ binary decisions must be performed to calculate this symbol. This is in fact a loose lower bound, but tells us that the user must perform at least $M(U-1)$ binary decisions per login.

If the output symbol does not change uniformly at random based on the U input bits things only get better for the attacker. Suppose that only one bit where $\mathbf{S}$ an $\mathbf{S}^{'}$ differ is used among the U that are used to calculate $R(t)$. Now if the probability that the symbol is unchanged is higher than a uniform assignment it merely serves to make the attacker's task easier, and the algorithm of Section 4.4 work better. Previously the attacker could infer closeness only when none of the e differing bits were involved. But now, when only one bit is involved the probability that $R(t) = R^{'}(t)$ is higher than when e bits are involved. Thus the probability that $|\mathbf{S} - \mathbf{S}^{'}|$ is small when

What is Needed to Resist Brute-Force? Since $M(U-1)$ is lower bound on the number of binary decisions the user must perform, we can decide the maximum permissible U for a given burden on the user. Unfortunately this is extremely low. If we assume the user can perform a single binary decision per second then, if a one minute login procedure is acceptable (*i.e.* it would take the user this time to respond to the challenge) then we have $U = 60/20 + 1 = 4$. Even this assumes that the user can reliably perform 60 binary decisions without error. Of course we cannot reduce $M = 20$, since we require a minimum of a 20-bit login.

We summarize the cost of brute-forcing a $U = 5$ scheme in Table 2. For each secret size N, and number of observed logins W we take the value of e from

Table 2. Time in minutes to brute-force a Challenge Response scheme for a given secret size and number of logins observed when $U = 5$ bits of the secret are involved in each output bit. This requires at least 80 binary decisions be made by the user, and a more than one minute login procedure.

W	$N = 50$	$N = 60$	$N = 70$	$N = 80$
10	9.9	24	16	58
15	10.5	15.9	23	32
20	12.2	20.5	30.2	42
25	17.5	27.8	41.4	57

Table 1 and calculate the complexity from (4). We assume that the attacker can perform 1000 evaluations per second for the $N = 50, U = 5, k = 2$ case when he has observed $W = 10$ logins. The costs are given in hours to have a 0.9375 probability of finding the secret.

We summarize the cost of brute-forcing such a scheme in Table 2. The costs are given in hours required to have a 0.9375 probability of finding the true secret using our brute force method. That is, by choosing the largest value of e for a given scheme we calculated the number of trials required using (4). We assumed estimated that 10000 trials per second could be performed. The table makes clear the necessity of using large secrets. We regard it as infeasible to expect the user to perform more than 60 binary decisions for a login, and thus secrets larger than $N = 80$ (*i.e.* the equivalent of a 24 digit PIN) must be used.

5 Conclusion

We have examined the question of whether "Something You Know" can be saved as the sole factor for authenticating a user in the presence of spyware. Our conclusion is negative. We find that in a challenge response scheme the number of bits U of the secret involved in each bit of the response is the key parameter to surviving brute force. Unfortunately the amount of binary decisions the user must perform increases at least linearly with this parameter. This gives a fundamental tradeoff for which there appear to be no good choices. The Weinshall scheme [17], which used about 7.8 bits of the secret per output bit required about 3 minutes for the user to respond to the challenge. If we try to limit the login to a one minute login procedure we find that given enough observed logins the scheme is quickly brute-forced. This is true independent of the details of the scheme.

Golle and Wagner [7] conclude that "something you know" schemes be tested with SatSolver. We would add that measuring the number of bits of the secret involved in each output bit is paramount. Unless U can be made large, brute-forcing is trivial. This suggests that good alternatives between passwords, which do withstand replay, and one-time password or two-factor schemes ar very hard to find.

References

1. http://www.rsasecurity.com
2. Herley, C., Florêncio, D.: How To Login From an Internet Café without Worrying about Keyloggers. In: Symp. on Usable Privacy and Security (2006)
3. Cheswick, W.: Johnny Can Obfuscate: Beyond Mother's Maiden Name. In: Proc. Usenix HotSec (2006)
4. Florêncio, D., Herley, C.: One-Time Password Access to Any Server Without Changing the Server. In: Wu, T.-C., Lei, C.-L., Rijmen, V., Lee, D.-T. (eds.) ISC 2008. LNCS, vol. 5222, pp. 401–420. Springer, Heidelberg (2008)
5. Florêncio, D., Herley, C.: KLASSP: Entering Passwords on a Spyware Infected Machine. In: Jesshope, C., Egan, C. (eds.) ACSAC 2006. LNCS, vol. 4186. Springer, Heidelberg (2006)
6. Florêncio, D., Herley, C., Coskun, B.: Do Strong Web Passwords Accomplish Anything? In: Proc. Usenix Hot Topics in Security (2007)
7. Golle, P., Wagner, D.: Cryptanalysis of a Cognitive Authentication Scheme. In: Symp. on Security and Privacy (2007)
8. Haller, N.: The S/KEY One-Time Password System. In: Proc. ISOC Symposium on Network and Distributed System Security (1994)
9. Herley, C., Florêncio, D.: Phishing as a Tragedy of the Commons. In: NSPW 2008, Lake Tahoe, CA (2008)
10. Jermyn, I., Mayer, A., Monrose, F., Reiter, M.K., Rubin, A.D.: The Design and Analysis of Graphical Passwords. In: Usenix Security (1999)
11. Lamport, L.: Password Authentication with Insecure Communication. Communications of the ACM (1981)
12. Lei, M., Xiao, Y., Vrbsky, S., Li, C.-C., Liu, L.: A Virtual Password Scheme to Protect Passwords. In: Proceedings of IEEE ICC (2008)
13. Lim, J.: Defeat spyware with anti-screen capture technology using visual persistence. In: SOUPS (2007)
14. Pashalidis, A., Mitchell, C.J.: Impostor: A single sign-on system for use from untrusted devices. In: Proceedings of IEEE Globecom (2004)
15. Pering, T., Sundar, M., Light, J., Want, R.: Photographic Authentication through Untrusted Terminals. IEEE Security and Privacy (2003)
16. Suo, X., Zuo, Y., Owen, G.S.: Graphical Passwords: a Survey. In: ACSAC (2005)
17. Weinshall, D.: Cognitive Authentication Schemes Safe Against Spyware. In: Symp. on Security and Privacy (2006)

A Breaking the Virtual Passwords Scheme of Lin *et al.* [12]

The recently suggested Virtual Passwords scheme of Lei *et al.* [12] unfortunately appears to fall to the divide-and-conquer attack described above. In that work, user secret is an n-digit PIN, such that $\mathbf{S} = S_0S_1...S_{n-1}$ where $S_i \in \{0, 1, .., 9\}$. For each login, server presents an n-digit challenge such that $\mathbf{C} = C_0C_1...C_{n-1}$ where $C_i \in \{0, 1, .., 9\}$. The response calculation function is defined as follows:

$$R(t) = \begin{cases} (aS_0 + C_0 + S_1 + b) \bmod Z & i = 0 \\ (aR(t-1) + C_i + S_i + b + S_{i+1}) \bmod Z & i = 1, 2, .., n-1 \end{cases} \quad (5)$$

where a and b are two other random numbers that user has to remember and a is relatively prime to Z in order to make the response function bijective.

In general, since $\mathbf{R}$ and $\mathbf{C}$ are observable by the attacker, the only unknown parameters for each response $R(t)$, are a, b, S_i and S_{i+1}. For the sake of usability, the response for each login is also an n-digit number , such that $R(t) \in \{0, 1, .., 9\}$. Therefore one has to set $Z = 10$, which also implies that $a \in \{1, 3, 7, 9\}$ due to relatively prime constraint. On the other hand, regardless of the actual b value, without loss of generality we can consider that $b \in \{0, 1, .., 9\}$ due to the properties of modular arithmetic. Therefore for each $R(t)$, attacker has to try $4 \times 10 \times 10 \times 10 = 4000$ different combinations of the parameters a, b, S_i and S_{i+1} to see which combination matches the observed response $R(t)$. After one login, the attacker is left only with $4000/10 = 400$ combinations as $R(t)$ can be only one of the 10 possible values. Similarly after second and third login, the attacker will have only 40 and 4 possible choices respectively. And finally, the attacker will get the true parameter combination after observing the fourth login.

In summary, in the worst case attacker reveals $\mathbf{S}$, a and b after observing four login sessions at a cost of $n(4000 + 400 + 40 + 4)$ trials. However, in fact she can do much better both in terms of the number of logins observed and the number of trials, since the same a and b is used for every $R(t)$ and a single $S(i)$ is used for multiple $R(t)$s.

B Explanatory Tables

Password Authentication
Client $\rightarrow$ Server: $\mathbf{U}, \mathbf{P}$
OTP Authentication
Client $\rightarrow$ Server: $\mathbf{U}, \mathbf{P}_i$
Challenge Response Authentication
Client $\rightarrow$ Server: $\mathbf{U}$
Client $\leftarrow$ Server: $\mathbf{C}_i$
Client $\rightarrow$ Server: $f(\mathbf{S}, \mathbf{C}_i)$

Fig. 6. The basic types of access control discussed: password, one-time password and challenge response

$\mathbf{S}$	N-bit secret shared by client and server
$\mathbf{R}$	M-bit client response for a single login
$R(t)$	k-bit symbol of the response ($\mathbf{R} = R(0)R(1)\cdots R(T-1)$)
$f()$	calculation performed by the user
$f_t()$	$R(t) = f_t(\mathbf{S}, \mathbf{C}_i)$
$\mathbf{\Gamma}$	Observed series of W logins ($\mathbf{\Gamma} = \mathbf{R}_0\mathbf{R}_1\cdots\mathbf{R}_{W-1}$)
U	# bits of $\mathbf{S}$ used to calculate each $R(t)$
$B_{pdf}(d, n, p)$	Binomial pdf for n trials with probability p.
$B_{cdf}(d, n, p)$	Binomial cdf for n trials with probability p.

Fig. 7. Summary of notation and symbols used

New Communication-Efficient Oblivious Transfer Protocols Based on Pairings

Helger Lipmaa

University College London, UK

Abstract. We construct two simple families of two-message $(n, 1)$-oblivious transfer protocols based on degree-t homomorphic cryptosystems with the communication of respectively $1 + \lceil n/t \rceil$ and $3 + \lceil n/(t+1) \rceil$ ciphertexts. The construction of both families relies on efficient cryptocomputable conditional disclosure of secret protocols; the way this is done may be of independent interest. The currently most interesting case $t = 2$ can be based on the Boneh-Goh-Nissim cryptosystem. As an important application, we show how to reduce the communication of virtually any existing oblivious transfer protocols by proposing a new related communication-efficient generic transformation from computationally-private information retrieval protocols to oblivious transfer protocols.

Keywords: Computationally-private information retrieval, conditional disclosure of secrets, homomorphic encryption, oblivious transfer.

1 Introduction

In an $(n, 1)$-oblivious transfer protocol, $(n, 1)$-OT, Alice on input $0 \leq \sigma < n$ retrieves the σth element of Bob's database $D = (D_0, \ldots, D_{n-1})$. One requires that Alice obtains no information about any D_j for $j \neq \sigma$, and that Bob obtains no information about σ. It is well-known that by general reductions, one can base both two-party computation [Yao82, IP07, Lip08] and multi-party computation [Kil88] on $(2, 1)$-OT. Efficient $(n, 1)$-OT is a cornerstone of many handcrafted cryptographic protocols. Thus, it is important to construct $(n, 1)$-OT protocols that are efficient for values of n ranging from $n = 2$ to say $n = 2^{20}$. The currently most communication-efficient $(n, 1)$-OT protocols for large n were proposed in [Lip05, GR05], while some of the most communication-efficient $(2, 1)$-OT protocols were proposed in [AIR01, LL07].

New Linear Protocols. We first propose two new families OTS_t and OTX_t, for $t \geq 1$, of linear-communication $(n, 1)$-OT protocols. Later in the paper we use these families to construct sublinear $(n, 1)$-OT protocols. Both families rely on a cryptosystem that enables to cryptocompute (that is, compute-on-ciphertexts) degree-t polynomials with coefficients from $\mathbb{Z}_N \cup \{\star\}$, where $\star$ denotes a pseudorandom element of the plaintext group $\mathbb{Z}_N$. (It's formally defined by multiplication and addition to elements of $\mathbb{Z}_N$.) We call such a cryptosystem *degree-t homomorphic*. The case $t = 1$ includes additively homomorphic cryptosystems like the Paillier [Pai99], and the case $t = 2$ includes the BGN cryptosystem [BGN05].

T.-C. Wu et al. (Eds.): ISC 2008, LNCS 5222, pp. 441–454, 2008.

Table 1. Comparison of different instantiations of OTX, OTS with the protocols from [AIR01, LL07]. Here, $|c|$ denotes the length of ciphertexts in bits; $|\mathsf{pk}|$ and $|c|$ depend on the underlying cryptosystem. Here '?' means that currently there are no known cryptosystems that are suitable in this case.

Protocol	Alice's comm.	Bob's comm.	Max ℓ	PKC	$\lvert c\rvert$	CDS eq.
Previous instantiations						
[AIR01] = OTS_1	$\lvert\mathsf{pk}\rvert + \lvert c\rvert$	$n\lvert c\rvert$	≤ 64	Mult. hom.	180	(7)
[LL07] = OTS_1	$\lvert\mathsf{pk}\rvert + \lvert c\rvert$	$n\lvert c\rvert$	≤ 680	Add. hom.	1536	(7)
New instantiations						
OTS_2	$\lvert\mathsf{pk}\rvert + \lvert c\rvert$	$\lceil n/2\rceil\lvert c\rvert$	≤ 64	BGN	1536	(7)
OTX_1	$\lvert\mathsf{pk}\rvert + 2\lvert c\rvert$	$n\lvert c\rvert$	≤ 680	Add. hom.	1536	(6)
OTX_2	$\lvert\mathsf{pk}\rvert + 3\lvert c\rvert$	$\lceil n/3\rceil\lvert c\rvert$	≤ 64	BGN	1536	(5)
Generic, hypothetical instantiations for $t > 2$						
OTS_t	$\lvert\mathsf{pk}\rvert + \lvert c\rvert$	$\lceil n/t\rceil\lvert c\rvert$	?	?	?	(7)
OTX_t	$\lvert\mathsf{pk}\rvert + 3\lvert c\rvert$	$\lceil n/(t+1)\rceil\lvert c\rvert$	?	?	?	(5)

Without loss of generality, assume that $t \mid n$. We also assume that the database elements are ℓ-bit long. Then, $(n,1)$-OTS_t is a parallel repetition of n/t copies of an atomic $(t,1)$-OTS_t protocol that use a common secret/public key pair. They also share Alice's first message that consists of the public key and of an encryption of Alice's index σ. In every single instance of $(t,1)$-OTS_t, Bob cryptocomputes his reply as a single encryption of the sum of two polynomials $\mathsf{Correct}_i^{t-1}(\sigma)$ and $\mathsf{CDSS}_i^t(\sigma)$, where the first polynomial takes care of the correctness and the second polynomial implements conditional disclosure of secrets (CDS, [GIKM00, AIR01, BGN05, LL07]) to guarantee Bob's privacy.

More precisely, $\mathsf{Correct}_i^t(\sigma)$ is the unique degree-t polynomial such that $\mathsf{Correct}_i^t(\sigma) = D_\sigma$ if $\lfloor \sigma/t \rfloor = i$, and $\mathsf{CDSS}_i^t(\sigma)$ is a degree-t polynomial such that $\mathsf{CDSS}_i^t(\sigma) = 0$ for $\lfloor \sigma/t \rfloor = i$ and $\mathsf{CDSS}_i^t(\sigma) = \star$ for $\lfloor \sigma/t \rfloor \neq i$. Thus, $\mathsf{Correct}_i^t(\sigma) + \mathsf{CDSS}_i^t(\sigma)$ is equal to D_σ if $\lfloor \sigma/t \rfloor = i$, and to $\star$, otherwise. In particular, OTS_1 corresponds to the $(n,1)$-OT protocols from [AIR01, LL07].

The protocol $(n,1)$-OTX_t is similarly composed from atomic $(t+1,1)$-OTX_t protocols. Here, however, Bob's reply is a sum of $\mathsf{Correct}_i^t(\sigma)$ and of a CDS polynomial $\mathsf{CDSX}'_i(\sigma)$ if $t = 1$, and of a CDS polynomial $\mathsf{CDSX}_i^t(\sigma)$ if $t > 1$. Because of the use of $\mathsf{Correct}_i^t(\sigma)$, the number of atomic protocols is decreased to $\lceil n/(t+1)\rceil$. However, the corresponding CDS polynomials are more complicated and require Bob to communicate 2 ciphertexts per *atomic* protocol (if $t = 1$), or Alice to communicate 3 ciphertexts (if $t > 1$). The basic reason behind the added complexity is that there is no degree-t polynomial f such that $f(\sigma) = 0$ for $\lfloor \sigma/(t+1) \rfloor = i$ and $f(\sigma) = \star$ for $\lfloor \sigma/(t+1) \rfloor \neq i$.

Given the state of the art on existing degree-t homomorphic cryptosystems and efficient CDS protocols, one can instantiate the protocols OTS_t and OTX_t with $t = 1$ or $t = 2$ as summarized in Table 1. (Here, the increase of $|c|$ to 1536 in factorization-based schemes takes into account the recent advances in factoring.) Thus, the new protocols are communication-efficient even when n is small, say $n = 2$ or $n = 3$. See Sect. 3 for more comparison.

New Sublinear Protocols. The most communication-efficient known sublinear $(n, 1)$-OT protocols are constructed by combining a communication-efficient $(n, 1)$-computationally-private information retrieval (CPIR) protocol such as [Lip05, GR05] with a linear $(n, 1)$-OT protocol from [AIR01, LL07], i.e., with OTS_1. For $\ell < 2^{64}$, the communication of the combined protocols decreases if OTS_1 is replaced with either OTS_2 or OTX_2. In the case of the only known CPIR protocol with log-communication [GR05], this replacement decreases slightly the communication of the combined protocol. In the case of Lipmaa's CPIR protocol from [Lip05], for small ℓ, the transformed oblivious transfer protocol is not only more secure but also more communication-efficient than Lipmaa's original CPIR protocol. We also point out that the existence of degree-2 cryptosystem with efficient decryption would imply the second log-communication oblivious transfer protocol.

General Remarks. Apart from presenting the concrete protocols, the current paper has a few more contributions. First, it provides a precise complexity analysis of the oblivious transfer protocols from [BGN05]. Second, it defines a clean methodology for cryptocomputing protocols, where Bob's answer is a sum of two polynomials, one of which takes care of the correctness and the second one takes care of Bob's privacy by using recent advances in defining efficient cryptocomputable protocols for conditional disclosure of secrets [LL07]. Third, it can be seen as a unification of several different oblivious transfers from the literature, and then generalisation to not yet studied cases.

Caveats. The proposed two-message protocols are secure only if the plaintext group order N of the underlying cryptosystem has no small prime divisors. This means that if the group order is composite (like in the case of existing additively homomorphic cryptosystems or the BGN cryptosystem) then one can either rely on the PKI model, use zero-knowledge proofs or correctness, or say use Lenstra's ECM algorithm to detect small divisors of N. See [LL07] for a discussion. This is not a problem if N is prime, for example, if we rely on lifted Elgamal. More relevantly, this is also not a problem if the cryptosystem does not have efficient decryption as it is the case with the BGN: in the case of BGN, one only has to verify that the smallest prime divisor p of N is large enough so that doing $O(\sqrt{p})$ operations is infeasible.

Notation. For a set S, $U(S)$ denotes the uniform distribution on it. $\star$ is used as a new element of some fixed group or ring, and is defined by it's multiplication or addition with other group elements. That is, if the group/ring order is prime, then $\star \cdot 0 = 0$, $\star \cdot i = \star$ and $\star + j = \star$ for any $i \neq 0$ and any j.

Road-map. In Sect. 2, we give necessary preliminaries. In Sect. 3, we describe the protocols OTS_t and OTX_t. In Sect. 4, we describe a generic transformation of any $(n, 1)$-CPIR protocol to a $(n, 1)$-OT protocol with a comparable communication. In Sect. 5, we discuss related work.

2 Preliminaries

Composite Order Bilinear Groups. Let $\mathbb{G}$ and $\mathbb{G}_T$ be two multiplicative cyclic groups of order N where $N = pq \in \mathbb{Z}$ and p, q are λ-bit primes for some fixed security

parameter $\lambda \in \mathbb{Z}^+$, $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ is a bilinear map, and for some fixed generator g of $\mathbb{G}$, $e(g,g)$ is a generator of $\mathbb{G}_T$. We assume that group operations and e are all efficiently computable. Let $\mathcal{G}$ be a bilinear group generation algorithm that outputs such a tuple $(p, q, \mathbb{G}, \mathbb{G}_T, e)$. [BGN05] suggest the following example. Pick large primes $p < q$ and let $N = pq$. Find the smallest ℓ so $P = \ell N - 1$ is prime and equal to 2 modulo 3. Consider the points on the elliptic curve $y^2 = x^3 + 1$ over $\mathbb{F}_P$. This curve has $P + 1 = \ell N$ points, so it has a subgroup $\mathbb{G}$ of order N. We let $\mathbb{G}_T$ be the order N subgroup of $\mathbb{F}^*_{P^2}$ and $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ be the modified Weil pairing from [BF03].

Let $(p, q, \mathbb{G}, \mathbb{G}_T, e) \leftarrow \mathcal{G}(\lambda)$. For an adversary $\mathcal{A}$, define $\mathsf{AdvSD}(\mathcal{A})$, the advantage of $\mathcal{A}$ in solving the *subgroup decision problem* [BGN05] as That is, the task of

$$\mathsf{AdvSD}_{(\mathbb{G},\mathbb{G}_T,e)}(\mathcal{A}) := |\Pr[x \leftarrow \mathbb{G} : \mathcal{A}(pq, \mathbb{G}, \mathbb{G}_T, e, x) = 1]| - |\Pr[x \leftarrow \mathbb{G} : \mathcal{A}(pq, \mathbb{G}, \mathbb{G}_T, e, x^q) = 1]| \ .$$

$\mathcal{A}$ is to distinguish random elements of $\mathbb{G}$ from random elements of its order p subgroup. We say that $(\mathbb{G}, \mathbb{G}_T, e)$ is a (τ, ε)-*SD group* if for any τ-time adversary $\mathcal{A}$, $\mathsf{AdvSD}_{(\mathbb{G},\mathbb{G}_T,e)}(\mathcal{A}) \le \varepsilon$.

Public-key Cryptosystems. A public-key cryptosystem is a tuple $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ of algorithms with (possibly public-key dependent) plaintext space $\mathcal{M}$, randomizer space $\mathcal{R}$ and ciphertext space $\mathcal{C}$, such that $\mathcal{G}$ generates a random secret/public key pair $(\mathsf{sk}, \mathsf{pk})$, $\mathcal{E}_{\mathsf{pk}}(m; r) = c$ encrypts a plaintext $m \in \mathcal{M}$ to a ciphertext $c \in \mathcal{C}$ by using a randomizer $r \in \mathcal{R}$, and $\mathcal{D}_{\mathsf{sk}}(c) = m$ decrypts a ciphertext $c \in \mathcal{C}$ to a plaintext $m \in \mathcal{M}$. One requires that for any $(\mathsf{sk}, \mathsf{pk}) \in \mathcal{G}$ and for any $m \in \mathcal{M}, r \in \mathcal{R}$, $\mathcal{D}_{\mathsf{sk}}(\mathcal{E}_{\mathsf{pk}}(m; r)) = m$. A public-key cryptosystem is (τ, ε)-*IND-CPA secure* if for a freshly generated public/secret key pair $(\mathsf{sk}, \mathsf{pk})$, any τ-time adversary $\mathcal{A}$ can distinguish random encryptions of any two plaintext messages m_1, m_2, even chosen by himself, with probability $\le \varepsilon$. (The probability is also taken over the choice of the keys.)

Additively Homomorphic Public-key Cryptosystems. A public-key cryptosystem is *additively homomorphic* if $\mathcal{M} = (\mathbb{Z}_N, +, 0)$ for some integer N, $(\mathcal{C}, \cdot, 1)$ is a finite cyclic group, and if

$$\mathcal{D}_{\mathsf{sk}}(\mathcal{E}_{\mathsf{pk}}(m_1; r_1) \cdot \mathcal{E}_{\mathsf{pk}}(m_2; r_2)) = m_1 + m_2$$

for any m_1, m_2, r_1, r_2. In addition, we require that $\mathcal{E}_{\mathsf{pk}}(m; r) \cdot \mathcal{E}_{\mathsf{pk}}(0; U(\mathcal{R})) = \mathcal{E}_{\mathsf{pk}}(m; U(\mathcal{R}))$ for any m, r; this enables to perform efficient rerandomization. There are many well-known additively homomorphic public-key cryptosystems, see for example, [Pai99, DJ01].

Disclose-if-equal. For an additively homomorphic cryptosystem, given an encryption $c = \mathcal{E}_{\mathsf{pk}}(m; r)$ of some m, one can compute $c_1 \leftarrow c^\star \cdot \mathcal{E}_{\mathsf{pk}}(0; U(\mathcal{R})) = \mathcal{E}_{\mathsf{pk}}(\star \cdot m; U(\mathcal{R}))$. If $\gcd(m, N) = 1$ (resp., $\gcd(m, N) > 1$) and $\star = U(\mathbb{Z}_N)$ then $c_1 = \mathcal{E}_{\mathsf{pk}}(U(\mathbb{Z}_N); U(\mathcal{R}))$ is a random encryption of a random value from $\mathbb{Z}_N$ (resp., in some nontrivial subgroup of $\mathbb{Z}_N$). In a *disclose-if-equal* protocol, Alice on input a obtains Bob's input b_1 if $a = b_2$ for Bob's second input b_2, otherwise Alice obtains $\star$. In a simple disclose-if-equal protocol [AIR01, LL07], given a random encryption of a, Bob computes a random encryption of

$$\star \cdot (b_2 - a) + b_1 \tag{1}$$

and returns it to Alice. However, this protocol is not secure by itself: if $b_2 - a$ is a non-trivial divisor of N, then because $\star \cdot (b_2 - a)$ belongs to a non-trivial subgroup of $\mathbb{Z}_N$, Alice can obtain partial information about b_1 [LL07]. This means that if decryption is inefficient, then this disclose-if-equal protocol is computationally private for Bob under the subgroup decision assumption. Otherwise, one should use the disclose-if-equal protocol of [LL07] that forces c_1 to be an encryption of a (statistically) pseudorandom value of $\mathbb{Z}_N$ for *any* $m \neq 0$, while c_1 is an encryption of 0 if $m = 0$. This can then used in the described disclose-if-equal protocol. Briefly, in the implementation of the Laur-Lipmaa protocol, instead of Eq. (1), one uses the polynomial

$$\star \cdot (b_2 - a) + \dagger \cdot 2^\ell + b_1 \ . \tag{2}$$

where $\dagger$ denotes the formal random element of $\mathbb{Z}_{\lfloor N/2^\ell \rfloor}$. Alice recovers the answers modulo 2^ℓ with $\ell < p - 1 - \varepsilon$, where p is the smallest prime divisor of N and $2^{-\varepsilon}$ is the desired privacy level of honest Bob. Denote by $\tilde{\mathbb{Z}}_N$ the set $\mathbb{Z}_N$ enhanced by all possible formal random elements that are computable by Bob. Thus, given an additively homomorphic cryptosystem, Bob can cryptocompute linear polynomials $f \in \tilde{\mathbb{Z}}_N[M_1, \ldots, M_t]$.

The BGN Cryptosystem and Degree-t Homomorphic Cryptosystems. The BGN cryptosystem is defined as follows [BGN05]. The algorithm $\mathcal{K}$ runs $\mathcal{G}$ to generate $(p, q, \mathbb{G}, \mathbb{G}_T, e)$. Let $N \leftarrow pq$. Pick generators $g, u \leftarrow U(\mathbb{G})$ and let $h \leftarrow u^q$. Output public key $\mathsf{pk} \leftarrow (N, \mathbb{G}, \mathbb{G}_T, e, g, h)$ and private key $\mathsf{sk} \leftarrow p$. To encrypt a message $m \in \mathbb{Z}_{2^\ell}$ where $2^\ell < q$ with public key pk, pick a random $r \leftarrow \mathcal{R} := \mathbb{Z}_N$ and compute $\mathcal{E}_{\mathsf{pk}}(m; r) \leftarrow g^m h^r \in \mathbb{G}$. To decrypt a ciphertext c using the private key sk, compute first $c^p = (g^m h^r)^p = (g^p)^m$ and then recover m by computing the discrete logarithm of c^p on base g^p. This can be done in time $O(2^{\ell/2})$ and thus one must take say $\ell < 64$ or $\ell = O(\log \lambda)$. Set $g_1 \leftarrow e(g, g)$ and $h_1 \leftarrow e(g, h)$, clearly g_1 has order N and h_1 has order q. Define the associated BGN cryptosystem $(\mathcal{E}^{\mathsf{a}}, \mathcal{D}^{\mathsf{a}})$ in group $\mathbb{G}_T$, with $\mathcal{E}^{\mathsf{a}}_{\mathsf{pk}}(m; r) := g_1^m h_1^r$ where $\mathcal{D}^{\mathsf{a}}$ is defined as the discrete logarithm of $\mathcal{E}^{\mathsf{a}}_{\mathsf{pk}}(m; r)^p$ on base g_1^p.

Given BGN encryptions of any m_1, m_2, one can compute a BGN encryption of $m_1 + m_2$ as $\mathcal{E}_{pk}(m_1) \cdot \mathcal{E}_{\mathsf{pk}}(m_2)$, and an associated BGN encryption of $m_1 m_2$ as $e(\mathcal{E}_{\mathsf{pk}}(m_1), \mathcal{E}_{\mathsf{pk}}(m_2))$. In particular,

$$\mathcal{E}^{\mathsf{a}}_{\mathsf{pk}}(m; r) = e(\mathcal{E}_{\mathsf{pk}}(m; r), g) \ .$$

Thus, given BGN encryptions of any $m_1, \ldots, m_t$, and using the disclose-if-equal protocol of Eq. 1, one can compute associated BGN encryptions of

$$\mathcal{E}^{\mathsf{a}}_{\mathsf{pk}}(f(m_1, \ldots, m_t)) \tag{3}$$

for any quadratic polynomial $f \in \tilde{\mathbb{Z}}_N[M_1, \ldots, M_t]$. This generalizes the computations that one can do in the case of additively homomorphic cryptosystems.

We call a cryptosystem *degree-t homomorphic* if one can cryptocompute (associated) encryptions of type Eq. (3) for any degree-t polynomial $f \in \tilde{\mathbb{Z}}_N[M_1, \ldots, M_t]$, given encryptions of M_i. Thus, $t = 1$ in the case of additively homomorphic public-key cryptosystems and $t = 2$ in the case of the BGN cryptosystem.

Conditional Disclosure of Secrets. During a conditional disclosure of secrets (CDS) protocol (see, for example, [GIKM00, AIR01, BGN05, LL07]), Alice obtains Bob's secret exactly iff her own input belongs to some publicly specified set of valid inputs; if Alice's input is incorrect then Alice obtains usually a value that is statistically close to a uniformly random plaintext. There exist several general approaches of constructing CDS protocols that are cryptocomputable given a degree-t homomorphic cryptosystem. In particular, efficient cryptocomputable CDS protocols for many tasks for $t = 1$ and $t = 2$ were respectively proposed in [AIR01, LL07] and [BGN05]; such protocols are usually based on disclose-if-equal subprotocols.

Oblivious Transfer. Assume that Alice has an input $\sigma \in \{0, \ldots, n-1\}$ and Bob has a database $D = (D_0, \ldots, D_{n-1})$ where $D_i \in \{0,1\}^\ell$. In an $(n,1)$-oblivious transfer protocol for ℓ-bit strings, $(n,1)$-OT$^\ell$, Alice obtains D_σ and no additional information, and Bob obtains no information about σ. We only consider two-message oblivious transfer (OT) protocols. An OT protocol is *correct* when in the case of honest parties, Alice receives D_σ. An OT protocol is (τ, ε_1)-*private for Alice* if for any two indices σ_1, σ_2, even chosen by Bob himself, a τ-time Bob cannot distinguish the first messages of Alice that correspond to σ_1, σ_2. An OT protocol is *statistically* ε_2*-private (resp., computationally* (τ_2, ε_2)*-private) for Bob* if there exists an unbounded simulator that, only given access to the first message of Alice and Bob's database element D_σ, generates Bob's second message from the distribution that is statistically ε_2-close to (resp., computationally (τ_2, ε_2)-indistinguishable from) Bob's response in the real protocol to Alice's first message. An OT protocol is *statistically (resp., computationally)* $(\tau_1, \varepsilon_1; \tau_2, \varepsilon_2)$*-relaxed-secure* if it is correct, (τ_1, ε_1)-private for Alice and statistically (resp., computationally) (τ_2, ε_2)-private for Bob. A statistically (resp., computationally) (τ, ε)-secure $(n,1)$-*computationally-private information retrieval (CPIR) protocol* is the same as a statistically (resp., computationally) $(\tau, \varepsilon; \mathrm{poly}(\lambda), 1)$-relaxed-secure OT protocol.

The presented security definition is standard in the case of CPIR and OT protocols [AIR01, Lip05, BGN05, NP05] but also say in the case of private keyword search protocols [FIPR05]. A proof that one can run many copies of corresponding protocols securely, while using the same public key in every copy, can be found in [LL07].

3 New Families of Oblivious Transfer Protocols

We next propose two families OTX_t and OTS_t of linear-communication $(n,1)$-OT protocols that use the properties of a degree-t cryptosystem to decrease the number of communicated ciphertexts to $3 + \lceil n/(t+1) \rceil$ and $1 + \lceil n/t \rceil$, respectively. Sect. 4 uses these linear protocols to construct sublinear protocols.

Underlying Idea of OTX_t. Without loss of generality, assume that $(t+1) \mid n$. The basic idea of the first new protocol, that we call $(n,1)$-OTX, follows. Alice first generates a new key pair for a degree-t homomorphic cryptosystem. She sends to Bob the new public key with a random encryption of σ. Given that, for every $0 \le i < n/(t+1)$, Bob cryptocomputes the polynomial $\mathsf{Correct}_i^t(\sigma) + \mathsf{CDSX}_i^t(\sigma)$, where $\mathsf{Correct}_i^t(\sigma)$ and $\mathsf{CDSX}_i^t(\sigma)$ are two degree-t polynomials that take care of protocol's correctness and

Bob's privacy respectively. More precisely, $\mathsf{Correct}_i^t$ is the unique degree-t polynomial, such that $\mathsf{Correct}_i^t(\sigma) = D_\sigma$ if $\lfloor \sigma/(t+1) \rfloor = i$. For example,

$$\begin{aligned}
\mathsf{Correct}_i^1(\sigma) =& ((2i+1)-\sigma)\cdot D_{2i} + (\sigma - 2i)\cdot D_{2i+1} \ , \\
\mathsf{Correct}_i^2(\sigma) =& \frac{1}{2}\cdot((3i+1)-\sigma)((3i+2)-\sigma)\cdot D_{3i} + \\
& (\sigma-3i)((3i+2)-\sigma)\cdot D_{3i+1} + \\
& \frac{1}{2}\cdot(\sigma-3i)(\sigma-(3i+1))\cdot D_{3i+2} \ .
\end{aligned}$$

Second, $\mathsf{CDSX}_i^t(\sigma)$ is a degree-t polynomial such that

$$\mathsf{CDSX}_i^t(\sigma) \begin{cases} 0 \ , & \lfloor \sigma/(t+1) \rfloor = i \ , \\ \mathsf{CDSX}_i^t(\sigma) = \star \ , & \text{otherwise} \ . \end{cases}$$

That is, CDSX_i^t implements a cryptocomputable conditional disclosure of secrets protocol. Therefore, $\mathsf{Correct}_i^t(\sigma) + \mathsf{CDSX}_i^t(\sigma)$ is equal to D_σ if $\lceil \sigma/(t+1) = i \rceil$, and to $\star$ otherwise.

A "minor" complication here is that such a polynomial CDSX must have degree $t+1$ while we need a degree-t polynomial. To overcome this issue, we let Alice send to Bob three encryptions of $(\sigma_2, \sigma_1, \sigma_0)$, where

$$\sigma_2 \leftarrow \lfloor \sigma/(t+1) \rfloor \ , \quad \sigma_1 \leftarrow \lfloor (\sigma \mod (t+1))/t \rfloor \ , \quad \sigma_0 \leftarrow \sigma \mod t \ . \tag{4}$$

E.g., if $\sigma = 14$ and $t = 4$ then $\sigma_2 = 2$, $\sigma_1 = 1$, and $\sigma_1 = 0$. From these encryptions, Bob can cryptocompute an encryption of $\sigma = (t+1)\sigma_2 + t\sigma_1 + \sigma_0$. We now redefine

$$\mathsf{CDSX}_i^t(\sigma_2,\sigma_1,\sigma_0) := \star\cdot(\sigma_2 - i) + \star\cdot(\sigma_1 - 1)\sigma_1 + \star\cdot\prod_{i=0}^{t-1}(\sigma_0 - i) + \star\cdot\sigma_1\sigma_0 \ . \tag{5}$$

Clearly, CDSX_i^t is a degree-t polynomial with the required properties, that is, $\mathsf{CDSX}_i^t(\sigma_2,\sigma_1,\sigma_0) = 0$ if $\lceil \sigma/(t+1) \rceil = i$ and $\mathsf{CDSX}_i^t(\sigma_2,\sigma_1,\sigma_0) = \star$, otherwise. (Here, the last 3 monomials together guarantee that the result is pseudorandom, unless $\sigma_0 \notin \{0,\ldots,t-1\}$ and $\sigma_1 = 0$, or $\sigma_0 = 0$ and $\sigma_1 = 1$, that is, unless $2\sigma_1 + \sigma_0 \notin \{0,\ldots,t\}$.)

After that, Bob returns all $n/(t+1)$ ciphertexts to Alice who decrypts the $\lfloor \sigma/(t+1) \rfloor$th ciphertext. Thus, if $0 \le \sigma < n$ then Alice retrieves D_σ, and if $\sigma \notin \{0,\ldots,n-1\}$ then Alice retrieves a close-to-uniformly random value.

The case $t = 1$ is different. In this case, we are not aware of a protocol with the communication of $\lceil n/2 \rceil + O(1)$ ciphertexts. The main problem is that the CDS protocol for showing that $x \in \{0,1\}$ by methods of [LL07] requires Bob to send *two* ciphertexts to Alice, because there is no way to check that $\sigma_0 \in \{0,1\}$ by using a single linear polynomial. Instead, as in [LL07], we transfer $\mathsf{Correct}_i^1$ twice, where the first time Alice obtains the answer if $\sigma_0 = 0$ and in the second time Alice obtains the answer if $\sigma_0 = 1$; this corresponds to the protocols of [AIR01, LL07]. More precisely, assume that $2 \mid n$. In OTX_1, Alice transfers to Bob one public key and two ciphertexts of $\sigma_1 = \lfloor \sigma/2 \rfloor$ and

$\sigma_0 = \sigma \mod 2$. For every $0 \le i < n/2$, Bob forwards to Alice random encryption of the vector $(\mathsf{Correct}_i^1(\sigma), \mathsf{Correct}_i^1(\sigma)) + \mathsf{CDSX}'_i(\sigma_1, \sigma_0)$, where

$$\mathsf{CDSX}'_i(\sigma_1, \sigma_0) := (\star \cdot (\sigma_1 - i) + \star \cdot \sigma_0, \star \cdot (\sigma_1 - i) + \star \cdot (\sigma_0 - 1)) \ . \tag{6}$$

Thus, the communication of OTX_1 is 1 public key and $n + 2$ ciphertexts.

Full Description of $(n, 1)$-OTX_2. We now follow up with a precise definition of the $(n, 1)$-OTX_t protocol. For simplicity's sake, we only give an implementation in the case $t = 2$ and assume that one uses the BGN cryptosystem. The general case is a straightforward extension.

Let $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ be the BGN cryptosystem with plaintext group order N; let p be the smallest prime divisor of N. Assume Alice's private input is $0 \le \sigma < n$ and Bob's private input is $D = (D_0, \ldots, D_{n-1})$. Fix $\ell < \log_2 p$ such that doing $O(2^{\ell/2})$ steps is feasible; for example, $\ell := 64$. (For the decryption to be polynomial-time in n, one needs that $\ell = O(\log n)$. However, in practical applications n is too small for the asymptotic notion to start to become relevant.) Without loss of generality, assume that $3 \mid n$. The protocol description follows:

1. Alice runs $\mathcal{K}$ to generate a new secret/public key pair $(\mathsf{sk}, \mathsf{pk})$. She stores sk. She computes $c_2 \leftarrow \mathcal{E}_{\mathsf{pk}}(\sigma_2; U(\mathcal{R}))$, $c_1 \leftarrow \mathcal{E}_{\mathsf{pk}}(\sigma_1; U(\mathcal{R}))$ and $c_0 \leftarrow \mathcal{E}_{\mathsf{pk}}(\sigma_0; U(\mathcal{R}))$, for σ_i computed according to Eq. (4), and sends $(\mathsf{pk}, c_2, c_1, c_0)$ to Bob.
2. If c_2, c_1 or c_0 is not a valid ciphertext then Bob rejects. Otherwise, Bob computes $c \leftarrow c_2^3 c_1^2 c_0$, $d_i \leftarrow \mathcal{E}_{\mathsf{pk}}(i; 0)$ for $i \in \{1, \ldots, n\}$, and a vector of ciphertexts $\boldsymbol{b} = (b_1, \ldots, b_{n/3})$, where

$$\begin{aligned} f_i \leftarrow & e(c_2/d_i, g)^{U(\mathbb{Z}_N)} \cdot e(c_1/d_1, c_1)^{U(\mathbb{Z}_N)} \cdot e(c_0/d_1, c_0)^{U(\mathbb{Z}_N)} \cdot e(c_1, c_0)^{U(\mathbb{Z}_N)} \ , \\ b_i \leftarrow & e(d_{3i-2}/a, d_{3i-1}/a)^{D_{3i}/2} \cdot e(a/d_{3i}, d_{3i-2}/a)^{D_{3i-1}} \cdot \\ & e(a/d_{3i}, a/d_{3i-1})^{D_{3i-2}/2} \cdot f_i \cdot h_1^{U(\mathcal{R})} \end{aligned}$$

   for $i \in \{1, \ldots, n/3\}$, and sends $\boldsymbol{b}$ to Alice.
3. Alice outputs $\mathcal{D}^{\mathsf{a}}_{\mathsf{pk}}(b_{\lfloor \sigma/3 \rfloor})$, or "reject" if decryption is not successful.

Theorem 1. *Assume that the BGN cryptosystem is $(\tau_{pkc}, \varepsilon_{pkc})$-IND-CPA secure, $(\mathbb{G}, \mathbb{G}_T, e)$ is a (τ_g, ε_g)-SD group, that the public key is correctly generated with $N = pq$ and $p < q$, and that $\ell = O(\log n) \ll \log_2 p$. Then the $(n, 1)$-OTX_2 protocol is computationally $(\tau_{pkc} - O(1), 3\varepsilon_{pkc}; \tau_g, \varepsilon_g)$-relaxed-secure.*

Proof.

CORRECTNESS: clearly, if c_j is generated correctly for $j \in \{0, 1, 2\}$, then b_i is a random associated encryption of a message distributed according to $X_i := \mathsf{Correct}_i^2(\sigma) + \mathsf{CDSX}_i^2(\sigma_2, \sigma_1, \sigma_0)$. Clearly, if $\sigma = 3\sigma_2 + 2\sigma_1 + \sigma_0 \in \{3i, 3i + 1, 3i + 2\}$ then $e = D_\sigma$.

ALICE'S PRIVACY: the only thing Bob sees is 3 ciphertexts (together with a fresh public key pk). Therefore, Alice's privacy follows directly from the IND-CPA security of the BGN cryptosystem.

BOB'S PRIVACY: we need to construct a simulator that on inputs $(\mathsf{pk}, D_\sigma, c_2, c_1, c_0)$ solely, where pk is a random public key and $\sigma \leftarrow \mathcal{D}_{\mathsf{sk}}(c_2^3 c_1^2 c_0)$, computes a second

round message that has almost the same distribution as b, that is, it is a random associated encryption of X_i. Simulator does the following. It rejects if any of c_i is not a valid ciphertext. First, if $\sigma \notin \{0, \ldots, n-1\}$, then it outputs a random associated encryption of a random element from $U(\mathbb{Z}_N)$. On the other hand, in this case, X_i is a random element of either $\mathbb{Z}_N$ or of some nontrivial subgroup of $\mathbb{Z}_N$ (e.g., when $\sigma_1 = p$). Thus, X_i and $U(\mathbb{Z}_N)$ are computationally (τ_g, ε_g)-indistinguishable by the subgroup decision assumption. Second, if $\sigma \in \{0, \ldots, n-1\}$ then the simulator outputs a random associated encryption of D_σ. Clearly, in this case simulator's output has distribution X_i. □

An Alternative Family OTS. We will next give a short description of an alternative family OTS of $(n, 1)$-OT$^\ell$ protocols. In OTS_t, Bob cryptocomputes polynomials

$$\mathsf{Correct}_i^{t-1}(\sigma) + \mathsf{CDSS}_i^t(\sigma) \ ,$$

where $\mathsf{Correct}_i^{t-1}$ is as defined before and CDSS_i^t is another, simpler, CDS polynomial. More precisely, assume that $t \mid n$. In OTS_t, Alice transfers a new public key and a random encryption of σ, and Bob replies with n/t random encryptions of $\mathsf{Correct}_i^t(\sigma) + \mathsf{CDSS}_i^t(\sigma)$, where

$$\mathsf{CDSS}_i^t(\sigma) := \star \cdot \prod_{j=0}^{t-1} (\sigma - (ti + j)) \tag{7}$$

for $0 \le i \le n/t - 1$.

Therefore, in OTS_t, Alice transfers 1 public key and 1 ciphertext, while Bob transfers $\lceil n/t \rceil$ ciphertexts (as opposed to 3 and $\lceil n/(t+1) \rceil$ ciphertexts in the case of OTX_t). Clearly, OTS_1 corresponds to the oblivious transfer protocol from [AIR01, LL07]. The only other current instantiation is OTS_2 when coupled with the BGN cryptosystem. To the best of our knowledge, if $\ell \le 64$ and one disregards the length of the public key and ciphertexts then OTS_2 is the most communication-efficient available $(2, 1)$-OT$^\ell$ protocol, having the total communication of 1 public key and 2 ciphertexts.

On the Use of Disclose-if-equal. Whenever the cryptosystem has efficient decryption, one must use the disclose-if-equal protocol of [LL07]. In this case, one must assume that $\ell < \log_2 p - \log_2 n - \varepsilon$, where $2^{-\varepsilon}$ is the desired statistical privacy-level of Bob.

Comparison. In the case $t = 1$, the underlying cryptosystem must be additively homomorphic. One can use either the lifted Elgamal (that has inefficient decryption) or say the Paillier [Pai99] or the Damgård-Jurik [DJ01]. Then, OTS_1 corresponds resp. to the Aiello-Ishai-Reingold protocol [AIR01] or to the Laur and Lipmaa protocol [LL07], while OTX_1 is a related but slightly less efficient protocol. Compared to the case $t = 2$, the case $t = 1$ benefits from the existence of a wide variety of additively homomorphic public-key cryptosystems, shorter public keys, and efficient decryption that makes it possible to obliviously transfer long strings with say $\ell \ge 680$. On the other hand, the number of transferred ciphertexts is larger than in the case of $t = 2$. Moreover, the ciphertexts of existing additively homomorphic cryptosystems are twice longer than the ciphertexts of the BGN cryptosystem. On the other hand, the ciphertexts of lifted elliptic-curve-based Elgamal are shorter than the ciphertexts of the BGN cryptosystem.

In the case $t = 2$, one uses a degree-2 homomorphic cryptosystem, for example, the Boneh-Goh-Nissim cryptosystem [BGN05]. Compared to $t = 1$, one now transfers less ciphertexts. Additionally, because these instantiations operate on the ciphertexts of the BGN cryptosystem, they can be used in conjunction with other protocols that rely on the BGN cryptosystem; such applications include efficient non-interactive zero-knowledge proofs from [GOS06]. On the other hand, one is currently restricted to the BGN cryptosystem that has longer public keys, compared to existing additively homomorphic public-key cryptosystems, and inefficient decryption that only allows to efficiently transfer strings with say $\ell \leq 64$.

From the communication-efficiency view-point, if neglecting the length of the public key and assuming that ℓ is small, for $n \leq 15$, the most efficient new protocol is $(n, 1)$-OTS_2, while for $n > 15$, the most efficient protocol is $(n, 1)$-OTX_2. In many common applications of oblivious transfer, the public key is shared with other protocols and thus does not incur a communication overhead.

Note that both $(n, 1)$-OTX and $(n, 1)$-OTS are secure only if one assumes that the public key is correctly generated. As in the case of protocols based on known additively homomorphic public-key cryptosystems, one needs that the smallest prime divisor of N is sufficiently large, see [LL07]. This assumption can be modeled by saying that this protocol is secure in the PKI model, or by letting Alice prove once in zero knowledge that the public key is correct and then using the same public key in many instances of the protocol. Yet another possibility is to use Lenstra's ECM algorithm to verify that N does not have small prime factors. These and other remedies are thoroughly discussed in [LL07]. In the case of the BGN, because it does not have efficient decryption, it is sufficient to verify that the smallest prime divisor p of N is larger than say 2^{160}.

4 Sublinear Oblivious Transfer

A common methodology to construct $(n, 1)$-OT protocols is to first construct a communication-efficient $(n, 1)$-CPIR protocol and then apply an efficient transformation to transfer it to a comparably efficient $(n, 1)$-OT protocol. Examples of communication-efficient $(n, 1)$-CPIR protocols include [Lip05, GR05]. A typical transformation was proposed in [AIR01] and later refined in [LL07] to work with existing additively homomorphic cryptosystems. Next, we generalize the approach of [AIR01, LL07].

We now describe a new transformation based on OTX_t for $t > 1$; the transformation based on OTS_t is similar. Without loss of generality, assume that $(t + 1) \mid n$. Recall that during the OTX_t protocol, Bob first constructs a database of $n/(t+1)$ ciphertexts, such that the ith ciphertext encrypts D_σ if $\lfloor \sigma/(t+1) \rfloor = i$, and $\star$, otherwise. Then Bob transfers the whole database of ciphertexts to Alice. Instead, we can use in parallel *any* two-message $(n/(t + 1), 1)$-CPIR protocol so that Alice will obtain the $\lfloor \sigma/(t + 1) \rfloor$th ciphertext. The resulting transformed protocol is clearly relaxed-secure: first, because OTX_t is relaxed-secure even if Alice sees *all* intermediate ciphertexts, the composed protocol is also relaxed-secure. Second, Bob only sees the first messages of Alice of both protocols and thus the composed protocols preserves Alice's privacy iff both OTX_t and the used CPIR protocol preserve Alice's privacy.

In general, let Π_1 be the OTX_t (or say the OTS_t) protocol, and let Π_2 be an arbitrary CPIR protocol. We denote the transformed protocol by $\Pi_2 \circ \Pi_1$, the case $\Pi_1 = \mathsf{OTS}_1$ corresponds to the transformation proposed in [AIR01, LL07]. Clearly, if Π_1 on database elements of length ℓ has the first message of $C_1(n,\ell)$ bits and the second message of $C_2(n,\ell)$ ciphertexts, and Π_2 on database elements of length λ with $C_3(n,\lambda)$ bits of communication, then the transformed protocol $\Pi_2 \circ \Pi_1$ has the communication of $C_1(n,\ell) + C_3(C_2(n,\ell),\lambda)$ bits. Here, λ is the length of ciphertexts in bits. Thus, $\Pi_2 \circ \mathsf{OTS}_1$ has the communication of $|\mathsf{pk}| + \lceil 2\log_2 N\rceil + C_3(n, \lceil 2\log_2 N\rceil)$ bits, where $|\mathsf{pk}| = \lceil \log_2 N\rceil \approx 1536$ bits. On the other hand, $\Pi_2 \circ \mathsf{OTX}_t$ has the communication of $|\mathsf{pk}| + 3\lceil \log_2 N\rceil + C_3(\lceil n/(t+1)\rceil, \lceil \log_2 N\rceil)$ bits, where $|\mathsf{pk}|$ is somewhat longer compared to the case of OTS_1.

If Π_2 is the Gentry-Ramzan CPIR protocol [GR05] with communication $O(\log_2 n + \ell)$ then the total communication of $\Pi_2 \circ \mathsf{OTS}_1$ is $|\mathsf{pk}| + O(\log_2 n + 2\log_2 N)$. In this case, the total communication of $\Pi_2 \circ \mathsf{OTX}_t$ is not significantly different unless t is large. On the other hand, the communication decrease is significant in the case of less communication-efficient CPIR protocols. Recall that Lipmaa's $(n,1)$-CPIR protocol [Lip05]—when used on top of the Damgård-Jurik cryptosystem [DJ01]—has the communication of

$$\left(\frac{1}{2}\cdot \log_2^2 n + (s+3/2)\cdot \log_2 n + s\right)\lambda$$

bits, where $\lambda = \lceil \log_2 N\rceil$, and s is the smallest integer such that $sk > \ell$ where k is the security parameter. Thus, applying Lipmaa's CPIR protocol on the OTX_2-transformed database of $n/3$ ciphertexts results in the protocol $\Pi_2 \cdot \mathsf{OTX}_2$ that has the communication of

$$\begin{aligned}&\left(3(s+1) + \frac{1}{2}\cdot \log_2^2 \frac{n}{3} + \left((s+1)+\frac{3}{2}\right)\cdot \log_2 \frac{n}{3} + (s+1)\right)\lambda \\ &\quad = \left(\frac{1}{2}\log_2^2 n + \left(s + \frac{5}{2} - \log_2 3\right)\log_2 n + (4-\log_2 3)\, s + 4 + \frac{5}{2}\cdot \log_2 3 + \right. \\ &\qquad \left. \frac{1}{2}\cdot \log_2^2 3\right)\lambda\end{aligned}$$

bits. This means that—assuming that the strings to be transferred are short with say $\ell \le 2^{64}$—the OTX_2-transformation actually *reduces* the communication of Lipmaa's original CPIR protocol, on top of increasing its security. This same will be true with virtually any superlogarithmic-communication CPIR protocol.

Recursive OTX_t. We can recursively apply OTX_t to itself. Bob's original database has n items, each ℓ bits. The intermediate database, generated by OTX_t has $\lceil n/(t+1)\rceil$ ciphertexts, each $\lceil \log_2 N\rceil$ bits. One can next apply the $(\lceil n/(t+1)\rceil, 1)$-$\mathsf{OTX}_t$ protocol $\xi := \lceil \log_2 N/\ell\rceil$ times to retrieve all $\lceil \log 2N\rceil$ bits of the $\lceil n/(t+1)\rceil$th intermediate ciphertext. Continuing, in the level r recursion, Alice sends 1 public key and $3r$ ciphertexts and Bob sends $\xi^{r-1}\cdot \lceil n/(t+1)^{r-1}\rceil$ ciphertexts.

Interestingly, if there existed a degree-2 homomorphic cryptosystem with $\xi = 2$ then this recursive construction would result in an $O(\log n)$ communication $(n,1)$-OT protocol. More precisely, $r \leftarrow (\ln n - \ln 6 + \ln\ln 1.5)/\ln 1.5$ would result in the optimal

communication of $(3\ln n+3-3\ln 6+3\ln\ln 1.5)/\ln 1.5 \approx 5.1\log_2 n-12.5$ ciphertexts. The same asymptotic result holds whenever $\xi \leq t$, while the optimal case for $\xi \geq t$ is just the trivial one with $r=1$.

5 Related Work

Boneh, Goh and Nissim [BGN05] considered the application of degree-2 homomorphic cryptosystems to construct efficient oblivious transfer protocols. They proposed two similar but yet different $(n,1)$-CPIR protocols. The next protocol is a symbiosis of both that achieves the same communication complexity as their second protocol but is somewhat simpler to execute. In addition, we provide the precise communication complexity estimate. In this protocol, $\ell = O(\log n)$ as in $(n,1)$-OTX. The database is viewed as comprising of $n^{1/3}$ chunks, each chunk containing $n^{2/3}$ entires, where Alice is interested in retrieving entry (I,J,K) of D. For $0 \leq i,j < \sqrt[3]{n}$, Alice sends Bob random encryptions of $[i=I]$ and $[j=J]$. Bob uses the encryption scheme's homomorphic properties to compute associated encryptions of

$$D_{I,J,k} = \sum_{0\leq i,j<\sqrt[3]{n}} [i=I][j=J]D_{i,j,k}$$

for $0 \leq k < \sqrt[3]{n}$. Bob sends the $\sqrt[3]{n}$ resulting associated ciphertexts to Alice who decrypts the Kth entry. As briefly mentioned in [BGN05], recursively applying this scheme results in a communication complexity $O(n^{\varepsilon}\lambda)$ for any $\varepsilon > \lambda$. More precisely, assuming that a ciphertext is $\eta\ell$ bits, after R rounds of recursion this protocol has the communication of $(2\lfloor 3^R/2\rfloor + \eta^{\lfloor 3^R/2\rfloor-1})n^{1/3^R}$ ciphertexts. In the asymptotically optimal case $3^R = \sqrt{2\log_\eta n}$, this results in the communication of $(1+o(1))\exp(\sqrt{2\ln\eta\cdot\ln n})$ ciphertexts. In the case of say $\eta = 24$ (for example, if ciphertexts are 1536 bits long and $\ell = 64$), this protocol is inferior to the protocol of Stern [Ste98].

The essential differences, compared to OTX_2, are: first, $(n,1)$-OTX_2 requires Alice to send three ciphertexts and Bob to send $\lceil n/3\rceil$ ciphertexts, while the protocols of [BGN05] that correspond to one-dimensional case require Alice to send n ciphertexts and Bob to send one ciphertext. Second, one can combine OTX_t and OTS_t with an arbitrary existing sublinear computationally-private information retrieval protocol to construct an almost as efficient oblivious transfer protocol. The oblivious transfer protocols from [BGN05] do not seem to share this property. In the case of protocols of [BGN05] it seems that one can only use standard communication-balancing techniques that are not in par with the state-of-the art CPIR protocols of [Lip05, GR05]. Third, the protocols from [BGN05] are not private for Bob, and thus one must couple them with say OTX_2 to design a real oblivious transfer protocol. In this sense, the new protocols are orthogonal to the protocols from [BGN05].

Open Problems. Constructing a degree-2 homomorphic cryptosystem with efficient decryption is a major open problem. As we showed in Sect. 4, such a cryptosystem would make it possible to construct another $(n,1)$-OT protocol with $O(\log n)$ communication. Constructing degree-t, for $t > 2$, homomorphic cryptosystems is another

well-known open problem. We stress that not much is known about degree-t, $t \geq 2$, homomorphic cryptosystems. It may come out that the ciphertext lengths of such cryptosystems grow linearly with t. A more specific open problem posed by this paper is to construct a degree-1 homomorphic cryptosystem based $(n, 1)$-OT protocol (for example, a more efficient version of OTX_1) with communication $O(1) + \lceil n/2 \rceil$.

Acknowledgments. We would like to thank Jens Groth and Brent Waters for helpful comments. The author was partially supported by the Estonian Science Foundation, grant 6848.

References

[AIR01] Aiello, W., Ishai, Y., Reingold, O.: Priced Oblivious Transfer: How to Sell Digital Goods. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 119–135. Springer, Heidelberg (2001)

[BF03] Boneh, D., Franklin, M.K.: Identity-Based Encryption from The Weil Pairing. SIAM Journal of Computing 32(3), 586–615 (2003)

[BGN05] Boneh, D., Goh, E.-J., Nissim, K.: Evaluating 2-DNF Formulas on Ciphertexts. In: Kilian [Kil05], pp. 325–341

[DJ01] Damgård, I., Jurik, M.: A Generalisation, A Simplification And Some Applications of Paillier's Probabilistic Public-Key System. In: Kim, K. (ed.) PKC 2001. LNCS, vol. 1992, pp. 119–136. Springer, Heidelberg (2001)

[FIPR05] Freedman, M.J., Ishai, Y., Pinkas, B., Reingold, O.: Keyword Search And Oblivious Pseudorandom Functions. In: Kilian [Kil05], pp. 303–324.

[GIKM00] Gertner, Y., Ishai, Y., Kushilevitz, E., Malkin, T.: Protecting Data Privacy in Private Information Retrieval Schemes. Journal of Computer and System Sciences 60(3), 592–629 (2000)

[GOS06] Groth, J., Ostrovsky, R., Sahai, A.: Perfect Non-Interactive Zero-Knowledge for NP. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 338–359. Springer, Heidelberg (2006)

[GR05] Gentry, C., Ramzan, Z.: Single-Database Private Information Retrieval with Constant Communication Rate. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 803–815. Springer, Heidelberg (2005)

[IP07] Ishai, Y., Paskin, A.: Evaluating Branching Programs on Encrypted Data. In: Vadhan, S. (ed.) TCC 2007. LNCS, vol. 4392, pp. 575–594. Springer, Heidelberg (2007)

[Kil88] Kilian, J.: Founding Cryptography on Oblivious Transfer. In: Proceedings of the Twentieth Annual ACM Symposiumon Theory of Computing, Chicago, Illinois, USA, 2-4 May 1988, pp. 20–31. ACM Press, New York (1988)

[Kil05] Kilian, J. (ed.): TCC 2005. LNCS, vol. 3378. Springer, Heidelberg (2005)

[Lip05] Lipmaa, H.: An Oblivious Transfer Protocol with Log-Squared Communication. In: Zhou, J., López, J., Deng, R.H., Bao, F. (eds.) ISC 2005. LNCS, vol. 3650, pp. 314–328. Springer, Heidelberg (2005)

[Lip08] Lipmaa, H.: Private Branching Programs: On Communication-Efficient Cryptocomputing. Technical Report 2008/107, International Association for Cryptologic Research (2008), http://eprint.iacr.org/2008/107

[LL07] Laur, S., Lipmaa, H.: A New Protocol for Conditional Disclosure of Secrets And Its Applications. In: Katz, J., Yung, M. (eds.) ACNS 2007. LNCS, vol. 4521, pp. 207–225. Springer, Heidelberg (2007)

[NP05] Naor, M., Pinkas, B.: Computationally Secure Oblivious Transfer. Journal of Cryptology 18(1), 1–35 (2005)

[Pai99] Paillier, P.: Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 223–238. Springer, Heidelberg (1999)

[Ste98] Stern, J.P.: A New And Efficient All Or Nothing Disclosure of Secrets Protocol. In: Ohta, K., Pei, D. (eds.) ASIACRYPT 1998. LNCS, vol. 1514, pp. 357–371. Springer, Heidelberg (1998)

[Yao82] Yao, A.C.-C.: Protocols for Secure Computations (Extended Abstract). In: 23rd Annual Symposium onFoundations of Computer Science, Chicago, Illinois, USA, 3–5 November 1982, pp. 160–164. IEEE Computer Society Press, Los Alamitos (1982)

A New (k, n)-Threshold Secret Sharing Scheme and Its Extension

Jun Kurihara, Shinsaku Kiyomoto, Kazuhide Fukushima, and Toshiaki Tanaka

KDDI R&D Laboratories, Inc.,
2-1-15 Ohara, Fujimino-Shi, Saitama 356-8502, Japan
{kurihara,kiyomoto,ka-fukushima,toshi}@kddilabs.jp
http://www.kddilabs.jp/

Abstract. In Shamir's (k, n)-threshold secret sharing scheme (threshold scheme), a heavy computational cost is required to make n shares and recover the secret. As a solution to this problem, several fast threshold schemes have been proposed. This paper proposes a new (k, n)-threshold scheme. For the purpose to realize high performance, the proposed scheme uses just EXCLUSIVE-OR(XOR) operations to make shares and recover the secret. We prove that the proposed scheme is a *perfect* secret sharing scheme, every combination of k or more participants can recover the secret, but every group of less than k participants cannot obtain any information about the secret. Moreover, we show that the proposed scheme is an *ideal* secret sharing scheme similar to Shamir's scheme, which is a *perfect* scheme such that every bit-size of shares equals that of the secret. We also evaluate the efficiency of the scheme, and show that our scheme realizes operations that are much faster than Shamir's. Furthermore, from the aspect of both computational cost and storage usage, we also introduce how to extend the proposed scheme to a new (k, L, n)-threshold *ramp* scheme similar to the existing *ramp* scheme based on Shamir's scheme.

Keywords: Secret sharing scheme, threshold scheme, threshold ramp scheme, exclusive-or, entropy, random number, ideal secret sharing scheme.

1 Introduction

A secret sharing scheme is an important tool for distributed file systems protected against data leakage and destruction, secure key management systems, etc. The basic idea of secret sharing introduced by Shamir and Blakley independently [1,2] is that a dealer distributes a piece of information (called a share) about the secret to each participant such that qualified subsets of participants can recover the secret but unqualified subsets of participants cannot obtain any information about the secret. Shamir's threshold scheme is based on polynomial interpolation ('Lagrange interpolation') to allow any k out of n participants to recover the secret.

T.-C. Wu et al. (Eds.): ISC 2008, LNCS 5222, pp. 455–470, 2008.

However, Shamir's scheme has two problems: large storage is required to retain all the shares, and heavy computational cost is needed to make shares and recover the secret due to processing a $(k-1)$-degree polynomial.

In order to reduce each bit-size of shares in Shamir's scheme, *ramp* secret sharing schemes have been proposed [3,4,5,6,7] that involve a trade-off between security and storage usage. In *ramp* schemes, we can consider *intermediate* sets, which are neither qualified nor forbidden sets to recover the secret, and hence, partially leak information on the secret. For instance, in the (k, L, n)-threshold *ramp* scheme[3,4], we can recover the secret from arbitrary k or more shares, but no information about the secret can be obtained from any $k-L$ or less shares. Furthermore, we can realize that every bit-size of shares is $1/L$ of the bit-size of the secret. However, an arbitrary set of $k-l$ shares is an *intermediate* set which leaks information about the secret with equivocation $(l/L)H(S)$ for $l = 1, 2, \ldots, L$, where S denotes the random variable induced by the secret s.

On the other hand, as a solution to the heavy computational cost problem associated with Shamir's scheme with no leak of information about the secret from $k-1$ or less shares, Ishizu et al. proposed a fast $(2, 3)$-threshold scheme[8]. By generalizing Ishizu et al.'s scheme for the number of participants, Fujii et al. introduced a fast $(2, n)$-threshold scheme[9,10]. These schemes enable fast computation to make shares and recover the secret from two or more shares by using just EXCLUSIVE-OR(XOR) operations. In these schemes, no information about the secret can be obtained from one share, but the secret can be recovered from each pair of shares. Furthermore, every bit-size of shares equals the bit-size of the secret as with Shamir's scheme. Especially, in Fujii et al.'s scheme, shares are constructed by concatenating XORed terms of a divided piece of the secret and a random number with the properties of prime numbers. These XORed terms are circulated in a specific pattern and do not overlap with each other. Kurihara et al. proposed a fast $(3, n)$-threshold scheme using XOR operations[11] as an extension of Fujii et al.'s scheme by constructing shares with the secret and two sets of random numbers, which are concatenated XORed terms of a divided piece of the secret and two random numbers. This $(3, n)$-threshold scheme is an *ideal* scheme as with Shamir's and Fujii et al.'s. Since no method has ever been investigated to extend the circulation property of this $(3, n)$-threshold scheme, an extension of this $(3, n)$-threshold scheme has not been proposed before.

Shiina et al. proposed another fast (k, n)-threshold scheme using XOR or additive operations[12]. This scheme can be applied to a cipher or signature which uses a homomorphism, and leaks no information about the secret from less than k shares. However, every bit-size of shares is $({}_nC_k - {}_{n-1}C_k) = O(n^{k-1})$ times as large as the bit-size of the secret. To address this efficiency problem, Kunii et al. introduced an alternative method[13] to construct shares in Shiina et al.'s scheme. However, the bit-size of shares is $\log_2 n$ or more times larger than the bit-size of the secret.

Thus, how to construct a fast (k, n)-threshold scheme using XOR operations such that every bit-size of shares equals the bit-size of the secret, where $k \geq 4$ and arbitrary n, remained an open question.

Our Contributions. In this paper, we present a new (k, n)-threshold scheme which realizes fast computation to make shares and recover the secret by using just XOR operations. Our contribution can be summarized as follows:

- We realize a new (k, n)-threshold scheme by constructing shares with the secret and $k-1$ sets of random numbers, which are concatenated XORed terms of a divided piece of the secret and $k-1$ random numbers. These XORed terms are circulated in a specific pattern with k dimensions, and do not overlap with each other because the properties of prime numbers are used.
- We show that the proposed scheme is a *perfect* secret sharing scheme, every combination of k or more participants can recover the secret, but every group of less than k participants cannot obtain any information about the secret. We also show that the proposed scheme is an *ideal* secret sharing scheme similar to Shamir's scheme, which is a *perfect* scheme such that every bit-size of shares equals that of the secret.
- By an implementation on a PC, we show that the proposed scheme is able to make n shares from the secret and recover the secret from k shares more quickly than Shamir's scheme if n is not extremely large. Under our implementation, our scheme performs the operations 900-fold faster than Shamir's for $(k, n) = (3, 11)$.
- We introduce how to extend our (k, n)-threshold scheme to a new (k, L, n)-threshold *ramp* scheme which realizes not only fast computation but also reduction of storage usage to retain n shares.

Organization. The rest of this paper is organized as follows: In Section 2, we give several notations and definitions, and provide a definition of the secret sharing scheme. In Section 3.1 of Section 3, we propose a new (k, n)-threshold scheme using just XOR operations. Moreover, in Section 3.2, we prove that our (k, n)-threshold scheme is an *ideal* secret sharing scheme as with Shamir's, and the efficiency of the proposed scheme is discussed in Section 4. In Section 5, we introduce how to extend our (k, n)-threshold scheme to a new (k, L, n)-threshold *ramp* scheme. Finally, we present our conclusions in Section 6.

2 Preliminaries

2.1 Notations and Definitions

Throughout this paper, we use the following notations and definitions:

- $\oplus$ denotes a bit-wise EXCLUSIVE-OR(XOR) operation.
- $\|$ denotes a concatenation of binary sequences.
- $n \in \mathbb{N}$ denotes the number of participants.
- n_p is a prime number such that $n_p \geq n$.
- Arithmetic operations $(\pm, \times)$ on values of indexes of random numbers, divided pieces of the secret, pieces of shares, their XORed terms, and their random variables, are performed modulo n_p. Hence, $X_{c(a \pm b)}$ denotes $X_{c(a \pm b) \bmod n_p}$.

– $H(X)$ denotes Shannon's entropy of a random variable X.
– $|\mathcal{X}|$ denotes the number of elements of a finite set $\mathcal{X}$.
– $2^{\mathcal{X}}$ denotes the family of all subsets of $\mathcal{X}$.

2.2 Secret Sharing Scheme

Let $\mathcal{P} = \{P_i \mid 0 \leq i \leq n-1,\ i \in \mathbb{N}_0\}$ be a set of n participants. Let $\mathcal{D}(\notin \mathcal{P})$ denote a dealer who selects a secret $s \in \mathcal{S}$ and gives a share $w_i \in \mathcal{W}_i$ to every participant $P_i \in \mathcal{P}$, where $\mathcal{S}$ denotes the set of secrets, and $\mathcal{W}_i$ denotes the set of possible shares that P_i might receive.

The access structure $\Gamma(\subset 2^{\mathcal{P}})$ is a family of subsets of $\mathcal{P}$ which contains the sets of participants qualified to recover the secret. Especially, Γ of a (k,n)-threshold scheme is defined by $\Gamma = \{A \in 2^{\mathcal{P}} \mid |A| \geq k\}$.

Let S and W_i be the random variables induced by s and w_i, respectively. A secret sharing scheme is *perfect* if

$$H(S|\mathcal{V}_A) = \begin{cases} 0 & (A \in \Gamma) \\ H(S) & (A \notin \Gamma) \end{cases}, \tag{1}$$

where $A \subset \mathcal{P}$ denotes a subset, and $\mathcal{V}_A = \{W_i \mid P_i \in A\}$ denotes the set of random variables of shares that are given to every participant $P_i \in A$. For any *perfect* secret sharing scheme, the inequation $H(S) \leq H(W_i)$ is satisfied[14,15].

Let $p(s)$ and $p(w_i)$ be the probability mass functions of S and W_i defined as $p(s) = \Pr\{S = s\}$ and $p(w_i) = \Pr\{W_i = w_i\}$, respectively. In general, the efficiency of a secret sharing scheme is measured by the information rate ρ [16] defined by $\rho = \dfrac{H(S)}{\max\limits_{P_i \in \mathcal{P}} H(W_i)}$. The maximum possible value of ρ equals one for *perfect* secret sharing schemes. When the probability distributions on $\mathcal{S}$ and $\mathcal{W}_i$ are uniform, i.e. $p(s) = 1/|\mathcal{S}|$ and $p(w_i) = 1/|\mathcal{W}_i|$, the information rate is $\rho = \dfrac{\log_2 |\mathcal{S}|}{\max\limits_{P_i \in \mathcal{P}} \log_2 |\mathcal{W}_i|}$, that is, the ratio between the length (bit-size) of the secret and the maximum length of the shares given to participants. A secret sharing scheme is said to be *ideal* if it is *perfect* and $\rho = 1$ [16,17,18]. Shamir's scheme[1] is recognized as being a typical *ideal* secret sharing scheme.

3 A (k,n)-Threshold Scheme

In this section, we describe the proposed (k,n)-threshold scheme. This scheme enables to make n shares (distribution) and recover the secret from k or more shares (recovery) using just XOR operations, for arbitrary threshold k and the number of participants n. We realize this scheme by extending the circulation property of Kurihara et al.'s $(3,n)$-threshold scheme[11]. Moreover, we show that our scheme is an *ideal* scheme as with Shamir's.

Table 1. Distribution Algorithm of Proposed (k, n)-Threshold Scheme

INPUT : $s \in \{0,1\}^{d(n_p-1)}$
OUTPUT : $(w_0, \ldots, w_{n-1})$

1: $s_0 \leftarrow 0^d$, $s_1 \parallel \cdots \parallel s_{n_p-1} \leftarrow s$
2: **for** $i \leftarrow 0$ to $k-2$ **do**
3: **for** $j \leftarrow 0$ to $n_p - 1$ **do**
4: $r_j^i \leftarrow GEN(\{0,1\}^d)$
5: **end for**
6: **end for** (discard $r_{n_p-1}^0$)
7: **for** $i \leftarrow 0$ to $n-1$ **do**
8: **for** $j \leftarrow 0$ to $n_p - 2$ **do**
9: $w_{(i,j)} \leftarrow \left(\bigoplus_{h=0}^{k-2} r_{h\cdot i+j}^h\right) \oplus s_{j-i}$
10: **end for**
11: $w_i \leftarrow w_{(i,0)} \parallel \cdots \parallel w_{(i,n_p-2)}$
12: **end for**
13: **return** $(w_0, \ldots, w_{n-1})$

Table 2. Recovery Algorithm of Proposed (k, n)-Threshold Scheme

INPUT : $(w_{t_0}, w_{t_1}, \ldots, w_{t_{k-1}})$
OUTPUT : s

1: **for** $i \leftarrow 0$ to $k-1$ **do**
2: $w_{(t_i,0)} \parallel \cdots \parallel w_{(t_i,n_p-2)} \leftarrow w_{t_i}$
3: **end for**
4: $\mathbf{w} \leftarrow (w_{(t_0,0)}, \ldots, w_{(t_0,n_p-2)}, \ldots, w_{(t_{k-1},0)}, \ldots, w_{(t_{k-1},n_p-2)})^{\mathrm{T}}$
5: $\mathbf{M} \leftarrow MAT(t_0, \ldots, t_{k-1})$
6: $(s_1, \ldots, s_{n_p-1})^{\mathrm{T}} \leftarrow \mathbf{M} \cdot \mathbf{w}$
7: $s \leftarrow s_1 \parallel \cdots \parallel s_{n_p-1}$
8: **return** s

3.1 Our Scheme

In this scheme, the secret $s \in \{0,1\}^{d(n_p-1)}$ needs to be divided equally into $n_p - 1$ blocks $s_1, s_2, \ldots s_{n_p-1} \in \{0,1\}^d$, where n_p is a prime number such that $n_p \geq n$, and $d > 0$ denotes the bit-size of every divided piece of the secret. Also, $\mathcal{D}$ uses n shares, $w_0, \cdots, w_{n-1}$, of a (k, n_p)-threshold scheme to construct a (k, n)-threshold scheme if the desired number of participants n is a composite number.

Table 1 and Table 4 denote the distribution algorithm and the structure of shares in our (k, n)-threshold scheme, respectively. To make shares, our (k, n)-threshold scheme requires 13 steps: First, $\mathcal{D}$ divides the secret $s \in \{0,1\}^{d(n_p-1)}$ into $n_p - 1$ pieces of d-bit sequence $s_1, \ldots, s_{n_p-1} \in \{0,1\}^d$ equally at step 1, where s_0 denotes a d-bit zero sequence, i.e. $s_0 = 0^d$ and $s_0 \oplus a = a$. We call this d-bit zero sequence a 'singular point' of divided pieces of the secret. [1] Next, at step 2-6, $(k-1)n_p - 1$ pieces of d-bit random number $r_0^0, \ldots, r_{n_p-2}^0$, $r_0^1, \ldots, r_{n_p-1}^1, \ldots, r_0^{k-2}, \ldots, r_{n_p-1}^{k-2}$ are chosen from $\{0,1\}^d$ independently from each other with uniform probability $1/2^d$, where $GEN(\mathcal{X})$ denotes a function to generate an $(\log_2 |\mathcal{X}|)$-bit random number from a finite set $\mathcal{X}$. At step 7-12, $\mathcal{D}$ makes pieces of shares by means of the following equation:

$$w_{(i,j)} = \left\{ \bigoplus_{h=0}^{k-2} r_{h\cdot i+j}^h \right\} \oplus s_{j-i}, \tag{2}$$

[1] It is not necessary for the singular point to be s_0, i.e. we can set an arbitrary singular point s_m $(0 \leq m \leq n_p - 1)$ and the others are $n_p - 1$ divided pieces of the secret. For the sake of simplicity, we suppose that the singular point is s_0 in this paper.

Table 3. Algorithm of the Function $MAT()$

INPUT : $t_0, t_1, \ldots, t_{k-1}$
OUTPUT : $\mathbf{M}$

1: **for** $i \leftarrow 0$ to $k-1$ **do**
2: **for** $j \leftarrow 0$ to $n_p - 2$ **do**
3: $\mathbf{v}_{(t_i,j)} \leftarrow VEC(t_i, j) = \left[\mathbf{i}_j^{n_p-1}\ \mathbf{i}_{t_i+j}^{n_p}\ \mathbf{i}_{2t_i+j}^{n_p}\ \cdots\ \mathbf{i}_{(k-2)t_i+j}^{n_p}\ \mathbf{i}_{j-t_i-1}^{n_p-1}\right]$
4: **end for**
5: **end for**
6: $\mathbf{G} \leftarrow (\mathbf{v}_{(t_0,0)}, \ldots, \mathbf{v}_{(t_{k-1},n_p-2)})^{\mathrm{T}}$
7: $\begin{bmatrix} \mathbf{G}_2 & \mathbf{G}_1 & \mathbf{J}_1 \\ \mathbf{\emptyset} & \mathbf{G}_0 & \mathbf{J}_0 \end{bmatrix} \leftarrow FG\left(\left[\mathbf{G} \quad \mathbf{I}_{k(n_p-1)}\right]\right) = \left[\bar{\mathbf{G}} \quad \mathbf{J}\right]$
8: $[\mathbf{I}_{n_p-1} \quad \mathbf{M}] \leftarrow BG\left([\mathbf{G}_0 \quad \mathbf{J}_0]\right)$
9: **return M**

Table 4. Structure of Shares of Proposed (k, n)-Threshold Scheme

	$j=0$	$j=1$	$\cdots$	$j=n_p-2$
$w_{(0,j)}$	$\left\{\bigoplus_{h=0}^{k-2} r_0^h\right\} \oplus s_0$	$\left\{\bigoplus_{h=0}^{k-2} r_1^h\right\} \oplus s_1$	$\cdots$	$\left\{\bigoplus_{h=0}^{k-2} r_{-2}^h\right\} \oplus s_{-2}$
$w_{(1,j)}$	$\left\{\bigoplus_{h=0}^{k-2} r_h^h\right\} \oplus s_{-1}$	$\left\{\bigoplus_{h=0}^{k-2} r_{h+1}^h\right\} \oplus s_0$	$\cdots$	$\left\{\bigoplus_{h=0}^{k-2} r_{h-2}^h\right\} \oplus s_{-3}$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$w_{(n-1,j)}$	$\left\{\bigoplus_{h=0}^{k-2} r_{h\cdot(n-1)}^h\right\} \oplus s_{-n+1}$	$\left\{\bigoplus_{h=0}^{k-2} r_{h\cdot(n-1)+1}^h\right\} \oplus s_{-n+2}$	$\cdots$	$\left\{\bigoplus_{h=0}^{k-2} r_{h\cdot(n-1)-2}^h\right\} \oplus s_{-n-1}$

where $0 \le i \le n-1, 0 \le j \le n_p-2$. Finally, $\mathcal{D}$ concatenates these pieces and constructs shares $w_i = w_{(i,0)} \parallel \cdots \parallel w_{(i,n_p-2)}$, and sends shares to each participant through a secure channel. If $n < n_p$, step 7-12 does not work for $0 \le i \le n_p - 1$ but it does for $0 \le i \le n-1$, and hence $\mathcal{D}$ does not generate $n_p - n$ shares $w_n, \cdots, w_{n_p-1}$. Thus, it is possible to add new participants $P_n, \cdots, P_{n_p-1}$ after distribution by generating $w_n, \cdots, w_{n_p-1}$ anew as necessary. However, to generate new shares, k existing shares should be gathered, and all random numbers and the secret should be stored.

Eq.(2) shows that these pieces of shares are circulated in a specific pattern with k dimensions by the indexes of a divided piece of the secret k random numbers, and do not overlap with each other because the properties of prime numbers are used.

Table 2 denotes the recovery algorithm in the scheme. First, each share is divided into d-bit pieces at step 1-3. Next, at step 4, $k(n_p-1)$-dimensional vector $\mathbf{w}$ is generated, which is a vector of divided pieces of shares. At step 5, $k(n_p-1) \times$

$k(n_p-1)$ binary matrix $\mathbf{M}$ is obtained by the function $MAT()$. All divided pieces of the secret, $s_1, \ldots, s_{n_p-1}$, are recovered by calculating $\mathbf{M} \cdot \mathbf{w}$ at step 6. Finally, the secret s is recovered by concatenating $s_1, \ldots, s_{n_p-1}$ at step 7.

Table 3 denotes the algorithm of the function $MAT()$ which makes the matrix $\mathbf{M}$. First, (kn_p-2)-dimensional binary vector $\mathbf{v}_{(t_i,j)}$ is obtained from indexes t_i and j at step 1-5. $VEC()$ denotes the function to make $\mathbf{v}_{(t_i,j)}$, where $\mathbf{i}_y^x$ denotes a x-dimentional binary row vector such that the only y-th element equals one $(0 \le y \le x-1)$ and the others are zero. $\mathbf{v}_{(t_i,j)}$ is defined as the generator vector of $w_{(t_i,j)}$, i.e. $w_{(t_i,j)} = \mathbf{v}_{(t_i,j)} \cdot \mathbf{r}$, where $\mathbf{r}$ is defined by

$$\mathbf{r} = (r_0^0,\ldots,r_{n_p-2}^0, r_0^1,\ldots,r_{n_p-1}^1, \ldots, r_0^{k-2},\ldots,r_{n_p-1}^{k-2}, s_1,\ldots,s_{n_p-1})^{\mathrm{T}},$$

where s_0 is omitted for the simple reason that $s_0 = 0^d$. For instance, $\mathbf{v}_{(0,1)} = (0100\ 01000\ 01000\ 1000)$ if $k = 4$ and $n_p = 5$. At step 6, the $k(n_p-1) \times (kn_p-2)$ binary matrix $\mathbf{G}$ is generated by $\mathbf{v}_{(t_0,0)}, \ldots, \mathbf{v}_{(t_{k-1},n_p-2)}$ as follows:

$$\mathbf{G} = \left(\mathbf{v}_{(t_0,0)}, \ldots, \mathbf{v}_{(t_0,n_p-2)}, \ldots, \mathbf{v}_{(t_{k-1},0)}, \ldots, \mathbf{v}_{(t_{k-1},n_p-2)}\right)^{\mathrm{T}},$$

which is the generator matrix such that $\mathbf{w} = \mathbf{G} \cdot \mathbf{r}$. At step 7, the matrix $[\mathbf{G} \quad \mathbf{I}_{k(n_p-1)}]$ is generated by column-wise concatenation, and transformed into a row echelon form $\left[\bar{\mathbf{G}} \quad \mathbf{J}\right] = FG\left(\left[\mathbf{G} \quad \mathbf{I}_{k(n_p-1)}\right]\right)$ by performing the forward elimination step of Gaussian elimination with the elementary row operations on GF(2), where $FG()$ and $\mathbf{I}_{k(n_p-1)}$ denote a forward elimination function and the $k(n_p-1) \times k(n_p-1)$ identity matrix, respectively. Furthermore, $\bar{\mathbf{G}}$ and $\mathbf{J}$ correspond to the transformed matrices from $\mathbf{G}$ and $\mathbf{I}_{k(n_p-1)}$, respectively. And, $\left[\bar{\mathbf{G}} \quad \mathbf{J}\right]$ is divided into block matrices denoted as follows:

$$\left[\bar{\mathbf{G}} \quad \mathbf{J}\right] = \begin{bmatrix} \mathbf{G}_2 & \mathbf{G}_1 & \mathbf{J}_1 \\ \mathbf{\emptyset} & \mathbf{G}_0 & \mathbf{J}_0 \end{bmatrix},$$

where $\mathbf{G}_0$, $\mathbf{G}_1$ and $\mathbf{G}_2$ are an $(n_p-1) \times (n_p-1)$ block matrix, $(k-1)(n_p-1) \times (n_p-1)$ block matrix and $(k-1)(n_p-1) \times (kn_p-n_p-1)$ block matrix, respectively. $\mathbf{J}_0$ and $\mathbf{J}_1$ are an $(n_p-1) \times k(n_p-1)$ block matrix and a $(k-1)(n_p-1) \times k(n_p-1)$ block matrix, respectively. $\mathbf{\emptyset}$ denotes a null matrix. Then, the backward substitution part of Gaussian elimination is executed on $[\mathbf{G}_0 \quad \mathbf{J}_0]$, and we obtain the matrix $\left[\mathbf{I}_{n_p-1} \quad \mathbf{M}\right] = BG([\mathbf{G}_0 \quad \mathbf{J}_0])$, where $BG()$ and $\mathbf{M}$ denote the function of backward substitution and a transformed matrix from $\mathbf{J}_0$, respectively. Finally, $MAT()$ outputs $\mathbf{M}$ as a matrix to recover $s_1, \ldots, s_{n_p-1}$ from divided pieces of shares.

Our (k, n)-threshold scheme proposed in this paper is a direct extension of Kurihara et al.'s $(3, n)$-threshold scheme[11] and Fujii et al.'s $(2, n)$-threshold scheme[9] in terms of the structure of shares. Accordingly, the distribution and recovery algorithms of our (k, n)-threshold scheme for $k = 3$ and $k = 2$ can be utilized as Kurihara et al.'s $(3, n)$-threshold scheme and Fujii et al.'s $(2, n)$-threshold scheme, respectively.

3.2 The Proof of the Ideal Secret Sharing Scheme

Here, we introduce the following two theorems.

Theorem 1. *Let A denote an arbitrary set of participants such that $|A| \leq k-1$. Then, since A is not in Γ of our proposed scheme, we have*

$$H(S|\mathcal{V}_A) = H(S), \tag{3}$$

where $\mathcal{V}_A$ denotes a set of random variables of shares that are given to each participant in A.

Proof (proof sketch). Let $A = \{P_{t_0}, \ldots, P_{t_{k-2}}\}$ denote a set of $k-1$ participants, where $t_0, \ldots, t_{k-2}$ are arbitrary numbers such that $0 \leq t_i, t_j \leq n-1$ and $t_i \neq t_j$ if $i \neq j$. Correspondingly, let $\mathcal{V}_A = \{W_{t_0}, \ldots, W_{t_{k-2}}\}$ denote a set of $k-1$ random variables, where $W_{t_0}, \ldots, W_{t_{k-2}}$ are induced by $w_{t_0}, \ldots, w_{t_{k-2}}$, respectively. And also, $W_{(t_i,0)}, \ldots, W_{(t_i,n_p-2)}$ denotes random variables induced by divided pieces of shares $w_{(t_i,0)}, \ldots, w_{(t_i,n_p-2)}$.

The following condition is supposed: $s_1, \ldots, s_{n_p-1}$, $r_0^0, \ldots, r_{n_p-2}^0$, $\ldots$, r_0^{k-2}, $\ldots$, $r_{n_p-1}^{k-2}$ are pairwise independent. And, $r_0^0, \ldots, r_{n_p-2}^0, \ldots, r_0^{k-2}, \ldots, r_{n_p-1}^{k-2}$ are chosen from the finite set $\{0,1\}^d$ with uniform probability $1/2^d$.

We define generator matrices $\mathbf{U}$ and $\mathbf{V}$ which satisfy the following equation:

$$\begin{aligned} \mathbf{W} &= \mathbf{U} \cdot \mathbf{R} \oplus \mathbf{V} \cdot \mathbf{S}, \\ &= (w_{(t_0,0)}, \ldots, w_{(t_0,n_p-2)}, \ldots, w_{(t_{k-2},0)}, \ldots, w_{(t_{k-2},n_p-2)})^{\mathrm{T}}, \end{aligned} \tag{4}$$

where $\mathbf{R}$ and $\mathbf{S}$ are denoted by $\mathbf{R} = (r_0^0, \ldots, r_{n_p-2}^0, r_0^1, \ldots, r_{n_p-1}^1, \ldots, r_0^{k-2}, \ldots, r_{n_p-1}^{k-2})^{\mathrm{T}}$ and $\mathbf{S} = (s_1, \ldots, s_{n_p-1})^{\mathrm{T}}$, respectively. $\mathbf{U}$ and $\mathbf{V}$ are $(k-1)(n_p-1) \times (kn_p-1)$ and $(k-1)(n_p-1) \times (n_p-1)$ matrices, respectively. Eq.(4) can be transformed into $\mathbf{U} \cdot \mathbf{R} = \mathbf{W} \oplus \mathbf{V} \cdot \mathbf{S}$. We consider the elementary row operation on $\mathbf{U}$ in this equation. Then, from Lemma 1, all rows of $\mathbf{U}$ are linearly independent. Hence, the hamming weight of each row of $\bar{\mathbf{U}}$ is one or more, where $\bar{\mathbf{U}}$ denotes a row-reduced echelon form of $\mathbf{U}$. Thus, each element of the vector obtained by $\bar{\mathbf{U}} \cdot \mathbf{R}$ is a random number or a XORed combination of $r_0^0, \ldots, r_{n_p-1}^{k-2}$. Therefore, all the elements of the vector obtained by $\bar{\mathbf{U}} \cdot \mathbf{R}$ are random numbers which are pairwise independent and uniformly distributed over $\{0,1\}^d$. This means that $\mathbf{W}$ obtained from any $\mathbf{S}$ is uniformly distributed over $\{0,1\}^{d(k-1)(n_p-1)}$. Thus, since $\mathbf{S}$ is independent from $\mathbf{W}$, we have $H(\mathbf{S}|\mathbf{W}) = H(\mathbf{S})$. Therefore, Eq.(3) is satisfied. □

Theorem 2. *Let A denote an arbitrary set of participants such that $|A| \geq k$. Then, since A is in Γ of our proposed scheme, the following equation is satisfied:*

$$H(S|\mathcal{V}_A) = 0. \tag{5}$$

where $\mathcal{V}_A$ denotes a set of random variables of shares that are given to each participant in A.

Proof (proof sketch). Let $t_0, \ldots, t_{k-1}$ be arbitrary numbers such that $0 \le t_i, t_j \le n-1$ and $t_i \ne t_j$ if $i \ne j$. Arithmetic operations $(\pm, \times)$ on values of indexes of random numbers, divided pieces of the secret, pieces of shares, their XORed terms, and their random variables are performed modulo n_p.

We define generator matrices $\mathbf{U}$ and $\mathbf{V}$ which satisfy the following equation:

$$\begin{aligned}\mathbf{W} &= \mathbf{U} \cdot \mathbf{R} \oplus \mathbf{V} \cdot \mathbf{S}, \\ &= (w_{(t_0,0)}, \ldots, w_{(t_0,n_p-2)}, \ldots, w_{(t_{k-1},0)}, \ldots, w_{(t_{k-1},n_p-2)})^{\mathrm{T}},\end{aligned}$$

where $\mathbf{R}$ and $\mathbf{S}$ are denoted by $\mathbf{R} = (r_0^0, \ldots, r_{n_p-2}^0, r_0^1, \ldots, r_{n_p-1}^1, \ldots, r_0^{k-2}, \ldots, r_{n_p-1}^{k-2})^{\mathrm{T}}$ and $\mathbf{S} = (s_1, \ldots, s_{n_p-1})^{\mathrm{T}}$, respectively. Let $\left[\bar{\mathbf{U}}\ \bar{\mathbf{V}}\right]$ denote the matrix transformed from $\left[\mathbf{U}\ \mathbf{V}\right]$ by the elementary row operation. Then, from Lemma 1, we can obtain the following vector by using XOR operations on divided pieces of shares:

$$\left[\bar{\mathbf{U}}\ \bar{\mathbf{V}}\right] \cdot \begin{bmatrix}\mathbf{R} \\ \mathbf{S}\end{bmatrix} = (*, \ldots, * \mid \{s_\alpha \oplus s_\beta\}, \{s_{\alpha+1} \oplus s_{\beta+1}\}, \ldots, \{s_{\alpha-2} \oplus s_{\beta-2}\})^{\mathrm{T}}.$$
$$\left(\alpha = t_{k-1} - t_{k-2} - \sum_{i=0}^{k-3} t_i, \qquad \beta = -t_{k-1} + t_{k-2} - \sum_{i=0}^{k-3} t_i\right).$$

Since n_p is a prime number, we can also obtain

$$s_{\alpha-1} \oplus s_{\beta-1} = \bigoplus_{m=0}^{n_p-2} (s_{\alpha+m} \oplus s_{\beta+m}).$$

Hence, we can consider the set $\mathcal{X} = \{x_m = s_{\alpha+m} \oplus s_{\beta+m} | 0 \le m \le n_p - 1\}$ to recover the secret. Then, since $\{pC = 2p(t_{k-2} - t_{k-1}) \mid 0 \le p \le n_p - 1\}$ is an additive group with order n_p,

$$\{C, 2C, \ldots, (n_p-1)C\} \equiv \{1, \ldots, n_p - 1\} \pmod{n_p}$$

is satisfied. Therefore, since $s_0 = 0^d$ was inserted as a singular point, we can recover all the divided pieces of the secret sequentially as follows:

$$\begin{aligned}
m = -\alpha \quad &: \quad s_C = x_{-\alpha}, \\
m = C - \alpha \quad &: \quad s_{2C} = x_{C-\alpha} \oplus s_C, \\
&\vdots \qquad\qquad \vdots \\
m = (n_p - 1)C - \alpha \quad &: \quad s_{(n_p-1)C} = x_{(n_p-1)C-\alpha} \oplus s_{(n_p-2)C},
\end{aligned}$$

Therefore, since all the divided pieces of the secret can be recovered from k shares, Eq.(5) is satisfied. □

From these two theorems, the access structure Γ of our scheme is denoted by $\Gamma = \{A \in 2^{\mathcal{P}} \mid |A| \ge k\}$, and Eq.(1) is satisfied. Therefore, our scheme is a *perfect* secret sharing scheme. Furthermore, since every bit-size of shares equals the bit-size of the secret if we can suppose that $s \in \{0,1\}^{d(n_p-1)}$ $(d > 0)$, i.e.

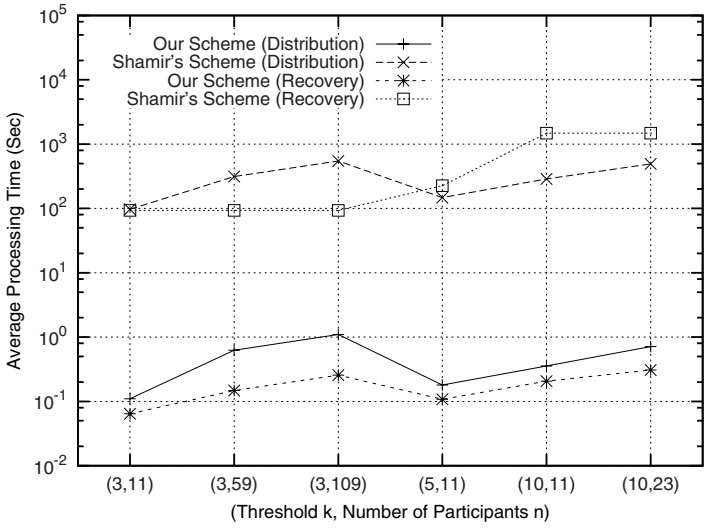


Fig. 1. Distribution and Recovery Processing Time for $n = n_p$

the size of the secret is $d(n_p - 1)$ bits,[2] the information rate ρ equals one. Thus, our scheme is *ideal* as with Shamir's.

4 Evaluation of Efficiency

In this section, we evaluate the efficiency of our scheme by comparing it with Shamir's scheme. First, we show the result of computer simulation by implementing both our scheme and Shamir's. Next, we consider the two schemes from the perspective of computational cost.

Computer Simulation. We compared the proposed scheme with that of Shamir's for $(k, n) = (3, 11)$, $(3, 59)$, $(3, 109)$, $(5, 11)$, $(10, 11)$ and $(10, 23)$ by implementation on a PC, where every scheme is implemented for $n = n_p$. Fig.1 denotes the processing time required to make $n(= n_p)$ shares from 4.5 MB data (secret) and recover the 4.5 MB secret from k shares, $w_0, \cdots, w_{k-1}$ by using our scheme and Shamir's scheme. The simulation environment and conditions are summarized in Table 5. For the implementation of Shamir's scheme, we used SSSS Version 0.5[19], which is a free software licensed under the GNU GPL. An 8-byte block was processed in each cycle in the distribution and recovery operations under Shamir's scheme. In Fig.1, the horizontal axis and vertical axis denote pairs of threshold and the number of participants, i.e. (k, n), and the processing time, respectively.

[2] If the size of the secret s were not multiple of $(n_p - 1)$, it is required to apply padding operations to the secret bit sequence to make shares and hence the bit-size of each share is larger than that of the secret.

Table 5. Simulation Environment and Conditions

CPU / RAM	: Pentium 4 3.0GHz / 2.0GB
Operating system	: Debian GNU/Linux 4.0
Compiler	: GCC 4.1
Source of random numbers	: /dev/urandom
Size of the secret s	: 4.5MB
(k, n)	: $(3, 11)$, $(3, 59)$, $(3, 109)$, $(5, 11)$, $(10, 11)$, $(10, 23)$
Implementation of Shamir's scheme	: SSSS Version 0.5[19]
Operating unit in Shamir's scheme	: 8 byte/operation

This graph shows that our scheme performed processing much faster than Shamir's. In $(3, 11)$-threshold schemes, our scheme was more than 900-fold faster than Shamir's in terms of both distribution and recovery. Similarly, in $(3, 59)$, $(3, 109)$, $(5, 11)$, $(10, 11)$ and $(10, 23)$-threshold schemes, Fig.1 shows that our scheme achieved far more rapid processing than Shamir's.

Consideration. In our proposed distribution algorithm, step 9 at Table 1 requires $(k-2)d$ bitwise XOR operations to make one divided piece of share $w_{(i,j)}$ which is constructed with s_0, or else, $(k-1)d$ bitwise XOR operations to make $w_{(i,j)}$ constructed without s_0. Thus, $(n_p-2)(k-1)d+(k-2)d$ XOR operations are required to make each share of $w_0, \ldots, w_{n_p-2}$. Furthermore, $(n_p-1)(k-1)d$ XOR operations are required to make w_{n_p-1}. Hence, the average number of XOR operations to make one share is $\left\{(k-1)-\frac{1}{n_p}\right\} \cdot \log_2 |\mathcal{S}|$. Therefore, our distribution algorithm requires an average of

$$\left\{(k-1)-\frac{1}{n_p}\right\} n \cdot \log_2 |\mathcal{S}| = O(kn) \cdot \log_2 |\mathcal{S}|,$$

bitwise XOR operations to make n shares. If $n = n_p$, it equals $\{(k-1)n - 1\} \cdot \log_2 |\mathcal{S}|$. Since the cost of modulo n_p operations on indexes can be regard as being negligible by using the fixed generator matrix in a manner similar to the recovery algorithm, we omit the cost of the operations here for the sake of simplicity.

On the other hand, in the proposed recovery algorithm, we can assume that at the most $\{k(n_p-1)-1\}d$ XOR operations are required to recover one of the divided pieces of the secret with all divided pieces of k shares, and at the most $\{k(n_p-1)-2\}d$ XOR operations are required to recover one of the other divided pieces of the secret with $k(n_p-1)-1$ divided pieces of k shares. Thus, the upper bound of the number of XOR operations required to recover the secret by using a block matrix $\mathbf{M}$ is roughly denoted by

$$\left\{k(n_p-1)-\frac{2n_p-3}{n_p-1}\right\} \cdot \log_2 |\mathcal{S}| = O(kn_p) \cdot \log_2 |\mathcal{S}|.$$

The recovery algorithm also requires $O(k^3 {n_p}^3)$ bitwise XOR operations to execute forward elimination (step 7 of Table 3) and partial backward substitution

(step 8 of Table 3) of Gaussian elimination as a pre-computation cost to obtain $\mathbf{M}$ at the function $MAT()$.

On the other hand, in Shamir's scheme, $O(kn)$ and $O(k \log^2 k)$ arithmetic operations are required to make shares and recover the secret, respectively [1].

From Fig.1, it is evident that the processing time for distribution in both Shamir's and our scheme is linearly increasing with each of k and n. However, though the processing time for recovery in Shamir's scheme is constant and independent of n if threshold k is fixed, that of our scheme increases as the number of participants $n(= n_p)$ grows in Fig.1. The computational cost of recovery in Shamir's scheme depends only on k, but that in our scheme depends on both k and n_p. Thus, though our scheme is much more efficient than Shamir's for not so large n_p as shown in Fig.1, our scheme will not perform faster processing to recover the secret than Shamir's if n_p is extremely large. We will determine the upper bound of n_p for the value of k as a future work, in which our scheme will be shown to be faster than Shamir's.

5 How to Extend Our Scheme to a Fast *Ramp* Scheme

In terms of improved efficiency for both computational cost and storage usage, a (k, L, n)-threshold *ramp* scheme [4,3] based on our (k, n)-threshold scheme can be realized. In this section, we briefly show how the new *ramp* scheme can be constructed.

In the distribution phase of our (k, L, n)-threshold *ramp* scheme ($1 \leq L \leq k-1$), the differences from our (k, n)-threshold scheme can be summarized as follows:

- The secret $s \in \{0,1\}^{dL(n_p-1)}$ is equally divided into $L(n_p-1)$ pieces $s_0^0, \ldots, s_{n_p-2}^0, \ldots, s_0^{L-1}, \ldots, s_{n_p-2}^{L-1} \in \{0,1\}^d$. And, the *singular points* in divided pieces of the secret are $s_{n_p-1}^0, \ldots, s_{n_p-1}^{L-1} = 0^d$.
- To make n shares, the dealer $\mathcal{D}$ generates $k-L$ sets of random numbers $\{r_0^0, \ldots, r_{n_p-2}^0\}$, $\{r_0^1, \ldots, r_{n_p-1}^1\}$, $\ldots$, $\{r_0^{k-L-1}, \ldots, r_{n_p-1}^{k-L-1}\}$, where the bit-size of each element in every set is d.
- The dealer makes pieces of shares $w_{(i,j)}$ by the following equation:

$$w_{(i,j)} = \left(\bigoplus_{h=0}^{k-L-1} r_{h \cdot i+j}^h \right) \oplus \left(\bigoplus_{h=0}^{L-1} s_{(k-L+h) \cdot i+j}^h \right).$$

The above differences mean that the *ramp* scheme can be realized by replacing $L-1$ sets of random numbers by an $L-1$ set of divided pieces of the secret, where each set of divided pieces of the secret has a singular point. On the other hand, we can recover the secret from k shares by similar recovery algorithm to our (k, n)-threshold scheme. The differences in the recovery phase are only the area of the generator matrix on which the partial backward substitution is performed and hence the size of matrix $\mathbf{M}$.

Then, the bit-size of each share is $1/L$ of the bit-size of the secret, and the efficiency in terms of computational cost for both distribution and recovery is

same as our (k, n)-threshold scheme: An average of $O(kn) \cdot \log_2 |\mathcal{S}|$ bitwise XOR operations is required to make n shares. To generate matrix $\mathbf{M}$, $O(k^3 n_p^3)$ bitwise XOR operations are required. Also, the upper bound of bitwise XOR operations to recover the secret by using $\mathbf{M}$ is $O(kn_p) \cdot \log_2 |\mathcal{S}|$.

In a manner similar to [4], it can be proved that the security property of this *ramp* scheme is same as Yamamoto's *ramp* scheme based on Shamir's scheme.

6 Conclusion

In this paper, we proposed a new (k, n)-threshold secret sharing scheme which uses just XOR operations to make shares and recover the secret, and we proved that the proposed scheme is an *ideal* secret sharing scheme. We estimated the computational cost in our scheme and Shamir's scheme for values of k and n. Also, we implemented our scheme on a PC for specific parameters, and showed that our scheme was more efficient than Shamir's in terms of computational cost provided n is not extremely large. Moreover, we introduced an extension of our scheme to a new (k, L, n)-threshold *ramp* scheme, which can realize both fast computation and reduction of storage usage.

References

1. Shamir, A.: How to share a secret. Commun. ACM 22(11), 612–613 (1979)
2. Blakley, G.R.: Safeguarding cryptographic keys. In: Proc. AFIPS, vol. 48, pp. 313–317 (1979)
3. Blakley, G.R., Meadows, C.: Security of ramp schemes. In: Blakely, G.R., Chaum, D. (eds.) CRYPTO 1984. LNCS, vol. 196, pp. 242–269. Springer, Heidelberg (1985)
4. Yamamoto, H.: On secret sharing systems using (k, L, n) threshold scheme. IEICE Trans. Fundamentals (Japanese Edition) J68-A(9), 945–952 (1985)
5. Kurosawa, K., Okada, K., Sakano, K., Ogata, W., Tsujii, T.: Non perfect secret sharing schemes and matroids. In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 126–141. Springer, Heidelberg (1994)
6. Ogata, W., Kurosawa, K.: Some basic properties of general nonperfect secret sharing schemes. J. Universal Computer Science 4(8), 690–704 (1998)
7. Okada, K., Kurosawa, K.: Lower bound on the size of shares of nonperfect secret sharing schemes. In: Safavi-Naini, R., Pieprzyk, J.P. (eds.) ASIACRYPT 1994. LNCS, vol. 917, pp. 34–41. Springer, Heidelberg (1995)
8. Ishizu, H., Ogihara, T.: A study on long-term storage of electronic data. In: Proc. IEICE General Conf., vol. D-9-10(1), p. 125 (2004) (in Japanese)
9. Fujii, Y., Tada, M., Hosaka, N., Tochikubo, K., Kato, T.: A fast $(2, n)$-threshold scheme and its application. In: Proc. CSS 2005, pp. 631–636 (2005) (in Japanese)
10. Hosaka, N., Tochikubo, K., Fujii, Y., Tada, M., Kato, T.: $(2, n)$-threshold secret sharing systems based on binary matrices. In: Proc. SCIS. pp. 2D1–4 (2007) (in Japanese)
11. Kurihara, J., Kiyomoto, S., Fukushima, K., Tanaka, T.: A fast $(3, n)$-threshold secret sharing scheme using exclusive-or operations. IEICE Trans. Fundamentals, E91-A(1), 127–138 (2008)

12. Shiina, N., Okamoto, T., Okamoto, E.: How to convert 1-out-of-n proof into k-out-of-n proof. In: Proc. SCIS 2004, pp. 1435–1440 (2004) (in Japanese)
13. Kunii, H., Tada, M.: A note on information rate for fast threshold schemes. In: Proc. CSS 2006, pp. 101–106 (2006) (in Japanese)
14. Karnin, E.D., Greene, J.W., Hellman, M.E.: On secret sharing systems. IEEE Trans. Inform. Theory 29(1), 35–41 (1983)
15. Capocelli, R.M., De Santis, A., Gargano, L., Vaccaro, U.: On the size of shares for secret sharing schemes. J. Cryptology 6, 35–41 (1983)
16. Blundo, C., De Santis, A., Gargano, L., Vaccaro, U.: On the information rate of secret sharing schemes. In: Brickell, E.F. (ed.) CRYPTO 1992. LNCS, vol. 740, pp. 149–169. Springer, Heidelberg (1993)
17. Stinson, D.R.: Decomposition constructions for secret sharing schemes. IEEE Trans. Inform. Theory 40(1), 118–125 (1994)
18. Stinson, D.R.: Cryptography: Theory and Practice. CRC Press, Florida (1995)
19. Poettering, B.: SSSS: Shamir's Secret Sharing Scheme, http://point-at-infinity.org/ssss/

Appendix 1: Lemma 1

In this appendix, we present Lemma 1, which shows the linear independence and dependence of rows of generator matrix $\mathbf{G}$. However, since the proof is too long to present in a paper because of the description about the elementary row operation on $\mathbf{G}$, we omit a detailed proof.

Lemma 1. *Let $t_0, \ldots, t_{L-1}$ denote indexes of $L-1$ shares, which are arbitrary numbers such that $0 \le t_i, t_j \le n-1$ and $t_i \ne t_j$ if $i \ne j$. Arithmetic operations $(\pm, \times)$ on values of indexes of matrices, vectors, random numbers, divided pieces of the secret, pieces of shares and their XORed terms are performed modulo n_p.*

Let the vectors $\mathbf{S}$ and $\mathbf{R}$ be denoted by $\mathbf{S} = (s_0, s_1, \ldots, s_{n_p-1})^{\mathrm{T}}$, and, $\mathbf{R} = (r_0^0, \ldots, r_{n_p-2}^0, r_0^1, \ldots, r_{n_p-1}^1, \ldots, r_0^{k-2}, \ldots, r_{n_p-1}^{k-2})^{\mathrm{T}}$, respectively. Let the matrices $\mathbf{U}$ and $\mathbf{V}$ be generator matrices of $L(n_p-1)$ pieces of L shares such that

$$\begin{aligned}\mathbf{W} &= \mathbf{U} \cdot \mathbf{R} \oplus \mathbf{V} \cdot \mathbf{S} \\ &= \left(w_{(t_0,0)}, \ldots, w_{(t_0,n_p-2)}, \ldots, w_{(t_{L-1},0)}, \ldots, w_{(t_{L-1},n_p-2)}\right)^{\mathrm{T}},\end{aligned}$$

where though $s_0 = 0^d$ is a singular point, we include s_0 as a variable in $\mathbf{S}$ to describe $\mathbf{V}$ briefly.

Then, the following equation is satisfied:

$$rank\left(\left[\mathbf{U}\ \mathbf{V}\right]\right) = \begin{cases} L(n_p-1)\ (1 \le L \le k-1) \\ k(n_p-1)\ (L \ge k) \end{cases}.$$

Remark 1. From Lemma 1, all rows of $\mathbf{U}$ are linearly independent if $1 \le L \le k-1$, and all rows of $\mathbf{U}$ are linearly dependent if $L \ge k$. Moreover, $\left[\mathbf{U}\ \mathbf{V}\right]$ can be transformed into $\left[\bar{\mathbf{U}}\ \bar{\mathbf{V}}\right]$ by the elementary row operation if $L = k$, which

satisfies the following equation:

$$\left[\bar{\mathbf{U}}\ \bar{\mathbf{V}}\right] \cdot \begin{bmatrix}\mathbf{R}\\ \mathbf{S}\end{bmatrix} = (*, \ldots, * \mid \{s_\alpha \oplus s_\beta\}, \{s_{\alpha+1} \oplus s_{\beta+1}\}, \ldots, \{s_{\alpha-2} \oplus s_{\beta-2}\})^{\mathrm{T}}. \quad (6)$$
$$\left(\alpha = -\sum_{i=0}^{k-3} t_i - t_{k-2} + t_{k-1}, \qquad \beta = -\sum_{i=0}^{k-3} t_i + t_{k-2} - t_{k-1}\right).$$

Proof (proof sketch). $\mathbf{U}$ and $\mathbf{V}$ can be denoted by

$$\mathbf{U} = \begin{pmatrix} \mathbf{I}_{n_p-1} & \mathbf{E}_{(t_0)} & \mathbf{E}_{(2t_0)} & \cdots & \mathbf{E}_{((k-2)t_0)} \\ \mathbf{I}_{n_p-1} & \mathbf{E}_{(t_1)} & \mathbf{E}_{(2t_1)} & \cdots & \mathbf{E}_{((k-2)t_1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{I}_{n_p-1} & \mathbf{E}_{(t_{L-1})} & \mathbf{E}_{(2t_{L-1})} & \cdots & \mathbf{E}_{((k-2)t_{L-1})} \end{pmatrix}, \qquad \mathbf{V} = \begin{pmatrix} \mathbf{E}_{((n_p-1)t_0)} \\ \mathbf{E}_{((n_p-1)t_1)} \\ \vdots \\ \mathbf{E}_{((n_p-1)t_{L-1})} \end{pmatrix},$$

respectively. $\mathbf{I}_{n_p-1}$ denotes an $(n_p-1) \times (n_p-1)$ identity matrix and $\mathbf{E}_{(j)}$ $(0 \le j \le n_p-1)$ denotes the following $(n_p-1) \times n_p$ matrix:

$$\mathbf{E}_{(j)} = \left(\begin{array}{ccc|c|ccc} 0 & \cdots & 0 & 0 & & & \\ \vdots & \ddots & \vdots & \vdots & & \mathbf{I}_{n_p-j} & \\ 0 & \cdots & 0 & 0 & & & \\ \hline & & & 0 & 0 & \cdots & 0 \\ & \mathbf{I}_{j-1} & & \vdots & \vdots & \ddots & \vdots \\ & & & 0 & 0 & \cdots & 0 \end{array}\right). \quad (7)$$

Then, by the elementary row operation on $[\mathbf{U}\ \ \mathbf{V}]$, we can obtain the following matrix $\mathbf{M}$ if $1 \le L \le k-1$:

$$\mathbf{M} = \begin{pmatrix} \mathbf{I}_{n_p-1} & \mathbf{E}_{(t_0)} & * & \cdots & * & * \cdots * & \mathbf{E}_{(-t_0)} \\ \mathbf{\emptyset} & \mathbf{E}^{(2)}_{(t_0,t_1)} & * & \cdots & * & * \cdots * & \mathbf{E}^{(2)}_{(-t_0,-t_1)} \\ \mathbf{\emptyset} & \mathbf{\emptyset} & \mathbf{E}^{(2)}_{(2t_1,2t_2)} & \cdots & * & * \cdots * & \mathbf{E}^{(2)}_{(f_2(t_2),g_2(t_2))} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \ddots \vdots & \vdots \\ \mathbf{\emptyset} & \mathbf{\emptyset} & \mathbf{\emptyset} & \cdots & \mathbf{E}^{(2)}_{(2t_{L-2},2t_{L-1})} & * \cdots * & \mathbf{E}^{(2)}_{(f_{L-1}(t_{L-1}),g_{L-1}(t_{L-1}))} \end{pmatrix},$$

where $\mathbf{E}^{(2)}_{(i,j)}$ denotes $\mathbf{E}^{(2)}_{(i,j)} = \mathbf{E}_{(i)} \oplus \mathbf{E}_{(j)}$. $f_m(t_i)$ and $g_m(t_i)$ are denoted by

$$f_m(t_i) = -\sum_{j=0}^{m-2} t_j - t_{m-1} + t_i, \quad g_m(t_i) = -\sum_{j=0}^{m-2} t_j + t_{m-1} - t_i,$$

respectively. Since the rank of $\mathbf{E}^{(2)}_{(i,j)}$ equals $n_p - 1$ if $i \not\equiv j \pmod{n_p}$, the rank of $\mathbf{M}$ equals $L(n_p-1)$ and all rows of $\mathbf{U}$ are linearly independent if $1 \le L \le k-1$.

In contrast, $[\mathbf{U} \;\; \mathbf{V}]$ can be transformed into the following matrix $\mathbf{M}$ if $L \geq k$:

$$\mathbf{M} = \begin{pmatrix} \mathbf{I}_{n_p-1} & \mathbf{E}_{(t_0)} & * & \cdots & * & \mathbf{E}_{((k-2)t_0)} \\ \emptyset & \mathbf{E}^{(2)}_{(t_0,t_1)} & * & \cdots & * & \mathbf{E}^{(2)}_{(-t_0,-t_1)} \\ \emptyset & \emptyset & \mathbf{E}^{(2)}_{(2t_1,2t_2)} & \cdots & * & \mathbf{E}^{(2)}_{(f_2(t_2),g_2(t_2))} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \emptyset & \emptyset & \emptyset & \cdots & \mathbf{E}^{(2)}_{(2t_{k-3},2t_{k-2})} & \mathbf{E}^{(2)}_{(f_{k-2}(t_{k-2}),g_{k-2}(t_{k-2}))} \\ \emptyset & \emptyset & \emptyset & \cdots & \emptyset & \mathbf{E}^{(2)}_{(f_{k-1}(t_{k-1}),g_{k-1}(t_{k-1}))} \\ \emptyset & \emptyset & \emptyset & \cdots & \emptyset & \emptyset \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \emptyset & \emptyset & \emptyset & \cdots & \emptyset & \emptyset \end{pmatrix}.$$

Thus, the rank of $\mathbf{M}$ equals $k(n_p - 1)$ and all rows of $\mathbf{U}$ are linearly dependent if $L > k$. Moreover, we can obtain the following vector with $\mathbf{M}$:

$$\mathbf{E}^{(2)}_{(f_{k-1}(t_{k-1}),g_{k-1}(t_{k-1}))} \cdot \mathbf{S} = \begin{pmatrix} (s_{f_{k-2}(t_{k-2})} \oplus s_{g_{k-2}(t_{k-2})}) \\ (s_{f_{k-2}(t_{k-2})+1} \oplus s_{g_{k-2}(t_{k-2})+1}) \\ \vdots \\ (s_{f_{k-2}(t_{k-2})-2} \oplus s_{g_{k-2}(t_{k-2})-2}) \end{pmatrix}.$$

Therefore, by the elementary row operation on $\left[\mathbf{U} \; \mathbf{V}\right]$, we can obtain the vector denoted at Eq.(6) if $L = k$. □

Appendix 2: A Short Example

We present a short description of the recovery procedure from w_0, w_1, w_2 and w_4 for $k = 4$ and $n = n_p = 5$ as an example. At step 5 of Table 2, we execute the function $MAT(0, 1, 2, 4)$ denoted at Table 3, and obtain 16×16 binary matrix $\mathbf{M}$. In the function $MAT()$, first, we obtain the generator matrix $\mathbf{G}$ from indexes of shares, which is denoted as follows:

$$\mathbf{G} = \begin{pmatrix} \mathbf{I}_4 & \mathbf{E}_{(0)} & \mathbf{E}_{(0)} & \bar{\mathbf{E}}_{(0)} \\ \mathbf{I}_4 & \mathbf{E}_{(1)} & \mathbf{E}_{(2)} & \bar{\mathbf{E}}_{(4)} \\ \mathbf{I}_4 & \mathbf{E}_{(2)} & \mathbf{E}_{(4)} & \bar{\mathbf{E}}_{(3)} \\ \mathbf{I}_4 & \mathbf{E}_{(4)} & \mathbf{E}_{(3)} & \bar{\mathbf{E}}_{(1)} \end{pmatrix}, \mathbf{E}_{(j)} = \left(\begin{array}{c} 0 \\ \vdots \\ 0 \\ \hline 1 \\ 0 \\ \vdots \\ 0 \end{array} \middle| \; \bar{\mathbf{E}}_{(j)} \right),$$

where $\mathbf{E}_{(j)}$ is the same matrix as Eq.(7). Then, by the elementary row operation on $[\mathbf{G} \;\; \mathbf{I}_{16}]$ in $MAT()$, $[\mathbf{I}_4 \;\; \mathbf{M}]$ is obtained, which is denoted as follows:

$$[\mathbf{I}_4 \quad \mathbf{M}] = \left(\begin{array}{c|c} 1000 & 1111\ 0001\ 0101\ 1011 \\ 0100 & 0111\ 1110\ 1000\ 0001 \\ 0010 & 0011\ 0110\ 0001\ 0100 \\ 0001 & 0001\ 0010\ 1010\ 1001 \end{array}\right).$$

At step 6 of Table 2, all divided pieces of the secret are recovered with $\mathbf{M}$ and $\mathbf{w}$ by the operation $(s_1, s_2, s_3, s_4)^{\mathrm{T}} = \mathbf{M} \cdot \mathbf{w}$.

Strong Accumulators from Collision-Resistant Hashing

Philippe Camacho[1,⋆], Alejandro Hevia[1,⋆⋆], Marcos Kiwi[2,⋆⋆⋆], and Roberto Opazo[3]

[1] Dept. of Computer Science, University of Chile, Blanco Encalada 2120, 3er piso, Santiago, Chile
{pcamacho,ahevia}@dcc.uchile.cl
[2] Dept. Ing. Matemática & Ctr. de Modelamiento Matemático, UMI 2807 U. Chile–CNRS
mkiwi@dim.uchile.cl
[3] CEO Acepta.com
roberto.opazo@acepta.com

Abstract. Accumulator schemes were introduced in order to represent a large set of values as one short value called the *accumulator*. These schemes allow one to generate membership proofs, i.e. short witnesses that a certain value belongs to the set. In universal accumulator schemes, efficient proofs of non-membership can also be created. Li, Li and Xue [11], building on the work of Camenisch and Lysyanskaya [5], proposed an efficient accumulator scheme which relies on a trusted accumulator manager. Specifically, a manager that correctly performs accumulator updates.

In this work we introduce the notion of *strong universal accumulator schemes* which are similar in functionality to universal accumulator schemes, but do not assume the accumulator manager is trusted. We also formalize the security requirements for such schemes. We then give a simple construction of a strong universal accumulator scheme which is provably secure under the assumption that collision-resistant hash functions exist. The weaker requirement on the accumulator manager comes at a price; our scheme is less efficient than known universal accumulator schemes — the size of (non)membership witnesses is logarithmic in the size of the accumulated set in contrast to constant in the scheme of Camenisch and Lysyanskaya.

Finally, we show how to use strong universal accumulators to solve a practical concern, the so called e-Invoice Factoring Problem.

Keywords: Accumulators, Collision-resistant Hashing, e-Invoice.

1 Introduction

Accumulator schemes were introduced by Benaloh and De Mare [3]. These primitives allow to represent a potentially very large set by a short value called *accumulator*. Moreover, the accumulator together with a so called *witness* provides an efficiently verifiable proof that a given element belongs to the accumulated set.

⋆ Gratefully acknowledges the support of CONICYT via FONDAP en Matemáticas Aplicadas.
⋆⋆ Gratefully acknowledges the support of CONICYT via FONDECYT No. 1070332.
⋆⋆⋆ Supported by CONICYT via FONDECYT No. 1010689 and FONDAP en Matemáticas Aplicadas.

T.-C. Wu et al. (Eds.): ISC 2008, LNCS 5222, pp. 471–486, 2008.

Barić and Pfitzmann [1] refined the security definition of accumulator schemes, and introduced the concept of collision-free accumulators. This notion was further extended by Camenisch and Lysyanskaya [5] to a dynamic setting where updates (additions and deletions) to the accumulator are possible. They proposed a new construction and showed how to use it to efficiently implement membership revocation in group signatures, and anonymous credential systems. In particular, they show how to keep track of valid identities using an accumulator, so proving membership is done by arguing in zero-knowledge that a certain secret value was accumulated. For a thorough discussion of accumulators we refer the interested reader to the survey of Fazio and Nicolosi [9].

Li, Li and Xue [11] recently introduced the notion of *universal accumulators*, which not only allow efficient generation of membership, but also of non-membership proofs. Building on [5], Li et al. construct universal accumulator schemes and point out useful applications, e.g. proving that a certificate has not been revoked, or that a patient does not have a disease. However, their construction inherits an undesirable property from Camenisch and Lysyanskaya's scheme; updates of the set (addition and deletion of elements) require the accumulator manager to be trusted. This falls short of Benaloh and De Mare's initial goal: to provide membership proofs even if the accumulator manager is corrupted.

We propose a new accumulator scheme based on hash trees similar to those used in the design of digital timestamping systems [3,2]. Recall that in hash trees values are associated to leaves of a binary tree. The values of sibling nodes are hashed in order to compute the value associated to their parent node, and so on and so forth, until a value for the root of the tree is obtained. The tree's root value is defined as the accumulator of the set of values associated to the leaves of the tree. We cannot directly use hash trees to obtain the functionality of universal and dynamic accumulators. Indeed, we need to add and delete elements from the accumulated set (tree node values if using hash trees) while at the same time be able to produce non–membership proofs. We solve this last issue using Kocher [10] trick; instead of associating values to the tree's leaves, we associated a pair of consecutive accumulated set elements. To prove that an element x is not in the accumulated set now amounts to showing that a pair (x_α, x_β), where $x_\alpha \prec x \prec x_\beta$, belongs to the tree but the pairs (x_α, x) and (x, x_β) do not.

The drawbacks of using a hash tree based scheme are twofold. First, the size of witnesses and the update time is logarithmic in the number of values accumulated. In contrast, witnesses and updates can be computed in constant time in RSA modular exponentiation based schemes like the ones of [5,3,1,11]. We believe, nonetheless, that this problem may in fact not exist for reasonable set sizes — a claim that we will later support. The second drawback is the accumulator's manager storage space requirements which is linear in the number of elements. However, this is not an issue for the accumulator's users, since they only need logarithmic in the accumulated set size storage space.

Overall, the main advantages of our scheme in comparison to Li et al.'s [11] are: (1) the accumulator manager need not to be trusted, and (2) since we only assume the existence of cryptographic hash functions as opposed to the Strong RSA Assumption, the underlying security assumption is (arguably) weaker. (Indeed, collision-resistance can be based on the intractability of factoring or computing discrete logarithms [7] while Strong-RSA is likely to be a stronger assumption than factoring [4].)

1.1 Our Contributions

Our contribution is threefold. First, we strengthen the basic definition of universal accumulators by allowing an adversary to corrupt the accumulator manager. This gives rise to the notion of *strong universal accumulators*. Second, we show how to construct strong universal accumulators using only collision-resistant hash functions. Our construction has interesting properties of its own. As in [5,11], we use auxiliary information to compute the (non)membership witness, but this information (called *memory*) need not to be kept private, and does not allow an adversary to prove inconsistent statements about the accumulated set. Indeed, the construction provides almost the same functionality as the (dynamic) universal accumulators described in [11], namely:

- All the elements of the set are accumulated in one short value.
- It is possible to add and remove elements from the accumulated set.
- For every element of the input space there exists a witness that proves whether the element has been accumulated or not.

Under stronger assumptions (concretely, Strong RSA) we show how to enhance our basic scheme in order to allow dynamic updates of witnesses.

Our last contribution is showing how to apply strong universal accumulators to solve a multi-party computational problem of practical relevance which we name the *e-Invoice Factoring Problem*. Solving this problem was indeed the original motivation that gave rise to this work.

In Section 2, we give some background definitions and formally introduce the notion of strong universal accumulator schemes. In Section 3, we describe our basic strong universal accumulator scheme and rigorously establish its security. In Section 4, we discuss the efficiency of the scheme in practice, and outline a variant that allows witness updates. Section 5 briefly motivates the e-Invoice Factoring Problem. In Section 6, we conclude with some comments. Due to space restrictions the e-Invoice Factoring Problem is described in the full version of this paper where it is also shown how it can be solved using strong universal accumulator schemes.

2 Definitions and Notations

Let $neg : \mathbb{N} \rightarrow \mathbb{N}$ denote a negligible function, that is, for every polynomial $p(\cdot)$ and any large enough integer n, $neg(n) < 1/p(n)$. Let also $||$ denote the operation of concatenation between binary strings. If $R()$ is a randomized algorithm, we write $a \xleftarrow{R} R()$ to denote the process of choosing a according to the probability distribution induced by R. We also denote by $\langle R() \rangle$ the set of all possible values a returned by R with positive probability. We distinguish between an *accumulator scheme* (the protocol, see below), its short representation or *accumulator value*, and its corresponding *accumulated set* X. For simplicity, however, we may use these terms indistinguishably when it's clear from the context.

SYNTAX. We formally define the syntax of a strong universal accumulator scheme (with memory). Our definition differs from that of Li et al. [11] as we consider an algorithm to verify if the accumulator value has been updated correctly (by adding or deleting a certain value), and we are not interested in hiding the order in which the elements are inserted into the accumulated set.

Definition 1 (Strong Universal Accumulators with Memory). *Let M be a set of values. A strong universal accumulator scheme (with memory) for the input set $X \subseteq M$ is a tuple* $\mathfrak{A} = (\mathsf{Setup}, \mathsf{Witness}, \mathsf{Belongs}, \mathsf{Update}, \mathsf{CheckUpdate})$ *where*

- Setup *is a randomized algorithm which on input a security parameter $k \in \mathbb{N}$, outputs a public data structure $\mathfrak{m}_0$ (also called the memory), and returns an initial accumulator value $\mathfrak{Acc}_0$ in the set $Y = \{0,1\}^k$. Both the accumulator value $\mathfrak{Acc}$ and the memory $\mathfrak{m}$ will be typically held and updated by the accumulator manager.*
- $\mathsf{Witness}$ *is a randomized algorithm which takes as input $x \in M$ and memory $\mathfrak{m}$, and outputs a witness of membership w if $x \in X$ (x has been accumulated) or a witness of nonmembership w' if $x \notin X$.*
- $\mathsf{Belongs}$ *is a randomized algorithm which on input a value $x \in M$, a witness w and the accumulator value $\mathfrak{Acc} \in Y$ outputs a bit 1 if w is deemed a valid witness that $x \in X$, outputs 0 if w is deemed a valid witness that $x \notin X$, or outputs the special symbol $\perp$ if w is not a valid witness of either statement.*
- $\mathsf{Update}_{\mathtt{op}}$ *is a randomized algorithm that updates the accumulator value by either adding an element ($\mathtt{op} = \mathtt{add}$) to or removing an element ($\mathtt{op} = \mathtt{del}$) from the accumulated set. The algorithm takes an element $x \in M$, an accumulator and memory pair $(\mathfrak{Acc}_{before}, \mathfrak{m}_{before})$, and outputs an updated accumulator and memory pair $(\mathfrak{Acc}_{after}, \mathfrak{m}_{after})$, and an update witness $w_{\mathtt{op}}$.*
- $\mathsf{CheckUpdate}$ *is a randomized algorithm that takes as input a value $x \in M$, a pair of accumulator values $(\mathfrak{Acc}_{before}, \mathfrak{Acc}_{after})$, and an update witness w, and returns a bit b. Typically, this algorithm will be executed by parties other than the accumulator manager in order to verify correct update of the accumulator by the manager. If $b = 1$, w is deemed a valid witness that an update operation (for $\mathtt{op} \in \{\mathtt{add}, \mathtt{del}\}$) which replaced $\mathfrak{Acc}_{before}$ with $\mathfrak{Acc}_{after}$ as the accumulator value, was valid. Otherwise, w is deemed invalid for the given accumulator pair.*

All the above algorithms are supposed to have complexity polynomial in the security parameter k.

In the above definition the memory $\mathfrak{m}$ is a public data structure which is computed from the set. Although public, this structure only needs to be maintained (stored) by the accumulator manager who updates the accumulator and generates membership and non-membership witnesses. In particular, the memory is *not* used to verify correct accumulator updates nor to check the validity of (non)membership witnesses.

Definition 2. *An accumulator value $\mathfrak{Acc}$ represents the set $X \subseteq M$, denoted by $\mathfrak{Acc} \Rightarrow X$, if and only if there exists a sequence $\{(\mathfrak{Acc}_i, x_i, \mathfrak{m}_i)\}_{1 \le i \le n}$, where $n = |X|$, and values $\mathfrak{Acc}_0, \mathfrak{m}_0$ where $x_i \in M$ for $1 \le i \le n$ and*

- $X = \{x_i\}_{1 \le i \le n}$,
- $(\mathfrak{Acc}_0, \mathfrak{m}_0) \in \langle \mathsf{Setup}() \rangle$,
- $(\mathfrak{Acc}_i, \mathfrak{m}_i, w_i) = \mathsf{Update}_{\mathtt{add}}(x_i, \mathfrak{Acc}_{i-1}, \mathfrak{m}_{i-1})$ *for all* $1 \le i \le n$.

If no such sequence exists $\mathfrak{Acc}$ does not represents set X, denoted by $\mathfrak{Acc} \not\Rightarrow X$.

Note that this definition also considers sets that have been formed by successive addition and deletions of elements as there is always a sequence of only addition operations that leads to the same set.

SECURITY. Universal accumulators as defined in [11] satisfy a basic consistency property: it must be unfeasible to find both a valid membership witness and a valid nonmembership witness for the same value $x \in M$. As mentioned there, this is equivalent to saying that given $X \subseteq M$ it is impossible to find $x \in X$ that has a valid nonmembership witness or to find $x \in M \backslash X$ that has a valid membership witness.

In order to be able to cope with malicious accumulator managers, we adapt the security notion in [11] as follows. First, we let the adversary select not only the value x and the witness w but also the accumulated set $X \subset Y$, the accumulator value $\mathfrak{Acc} \in Y$ and whether x belongs or not to X. We restrict the adversary so he must choose a pair $(\mathfrak{Acc}, X)$ for which there exists a sequence of valid addition operations (namely, $\mathsf{Update}_{\mathsf{add}}$ with values in X) that can produce an accumulated value $\mathfrak{Acc}$. This last restriction can be justified by noticing that, in the scenario we consider, parties other than the accumulator manager can externally verify the correctness of each update operation by using the CheckUpdate algorithm. Thus, security holds as long as it is unfeasible for the adversary to fool the CheckUpdate verification, namely given an accumulator value $\mathfrak{Acc}_{before}$, the adversary is unable to efficiently generate an accumulator value $\mathfrak{Acc}_{after}$, a set X, an input value x, and a valid update witness w for which $\mathfrak{Acc}_{before}$ actually represents set X and $\mathsf{CheckUpdate}(x, \mathfrak{Acc}_{before}, \mathfrak{Acc}_{after}, w) = 1$, but $\mathfrak{Acc}_{after} \not\Rightarrow X \cup \{x\}$ if w is an addition witness or $\mathfrak{Acc}_{after} \not\Rightarrow X \setminus \{x\}$ if w is an deletion witness.

Definition 3 (Security of Strong Universal Accumulators with Memory). *A strong universal accumulator with memory is secure if for every probabilistic polynomial-time adversary $\mathcal{A}$ the following conditions hold:*

- *(Consistency)*

$$Pr\left[\begin{array}{c}(x, w_1, w_2, X, \mathfrak{Acc}) \leftarrow \mathcal{A}(k);\\ \mathfrak{Acc} \Rightarrow X,\ \mathsf{Belongs}(x, w_1, \mathfrak{Acc}) = 1,\ \mathsf{Belongs}(x, w_2, \mathfrak{Acc}) = 0\end{array}\right] = neg(k).$$

- *(Secure addition)*

$$Pr\left[\begin{array}{c}(\mathfrak{Acc}_{before}, X, \mathfrak{Acc}_{after}, x, w) \leftarrow \mathcal{A}(k):\\ \mathfrak{Acc}_{before} \Rightarrow X,\ \mathfrak{Acc}_{after} \not\Rightarrow X \cup \{x\},\\ \mathsf{CheckUpdate}(x, \mathfrak{Acc}_{before}, \mathfrak{Acc}_{after}, w) = 1\end{array}\right] = neg(k).$$

- *(Secure deletion)*

$$Pr\left[\begin{array}{c}(\mathfrak{Acc}_{before}, X, \mathfrak{Acc}_{after}, x, w) \leftarrow \mathcal{A}(k):\\ \mathfrak{Acc}_{before} \Rightarrow X,\ \mathfrak{Acc}_{after} \not\Rightarrow X \backslash \{x\},\\ \mathsf{CheckUpdate}(x, \mathfrak{Acc}_{before}, \mathfrak{Acc}_{after}, w) = 1\end{array}\right] = neg(k).$$

The type of accumulators we consider in this work is not necessarily *quasi-commutative* [5,11] as they may not hide the order in which the elements were added to the set. More precisely, our definition tolerates that the value of the accumulator may depend on a particular sequence of $\mathsf{Update}_{\mathsf{add}}$ and $\mathsf{Update}_{\mathsf{del}}$ operations that produced a particular

accumulator value $\mathfrak{Acc}$. Our only requirement is that the accumulated set X represented by any accumulator value is well defined. The following proposition shows this is so if we use a secure strong universal accumulator scheme. The proof is omitted due to space constraints.

Proposition 1. *Let $\mathfrak{A}$ a secure strong universal accumulator scheme, and $k \in \mathbb{N}$ a security parameter. Given any adversary $\mathcal{A}$, consider the experiment $Exp^{SUAcc}_{\mathfrak{A},\mathcal{A}}$ in which the adversary is allowed to submit as many queries to oracle $O()$ as it wants and then stops. Oracle $O()$ is stateful and operates as follows: on any first query, the oracle creates an empty set X', runs $\mathsf{Setup}(k)$ to obtain $(\mathfrak{Acc}', \mathfrak{m}')$ which it returns as the query answer. Then, for each subsequent query of the form $(x, \mathfrak{Acc}, w)$ the oracle computes $b \leftarrow \mathsf{CheckUpdate}(x, \mathfrak{Acc}', \mathfrak{Acc}, w)$, and if $b = 1$, it sets $\mathfrak{Acc}' \leftarrow \mathfrak{Acc}$, $X' \leftarrow X' \cup \{x\}$, and returns bit b as the answer to the oracle query. If $b = 0$ it does not modify $\mathfrak{Acc}$ or X' and it simply returns $\perp$. We say adversary $\mathcal{A}$ wins $Exp^{SUAcc}_{\mathfrak{A},\mathcal{A}}$ if after $\mathcal{A}$ stops, it holds that $\mathfrak{Acc} \not\Rightarrow X'$. Then, for every probabilistic polynomial time adversary $\mathcal{A}$,* $\Pr\left[\mathcal{A} \text{ wins in } Exp^{SUAcc}_{\mathfrak{A},\mathcal{A}}\right]$ *is negligible in k.*

Our security definition (Definition 3) for the dynamic scenario (where addition and deletion of elements are allowed) differs from the one in [5] where the adversary is only able to add and delete elements by querying the accumulator manager, who is uncorruptible. In contrast, in our definition the adversary is allowed to control the accumulator. However, we require that during each update at least an uncorrupted participant verifies the update with $\mathsf{CheckUpdate}$ to guarantee the consistency between the accumulated value and the history of additions and deletions.

DYNAMIC ACCUMULATORS. The standard definition of dynamic accumulators (see for example the one in [5]) adds two requirements which so far we have not considered. First, it requires the existence of an additional efficient algorithm that allows to publicly and efficiently update membership witnesses after a change in the accumulator value so witnesses can be proven valid under the new accumulator value. And secondly, it requires that both the accumulator updating algorithm as well as the witness updating algorithm to run in time independent from the size n of the accumulated set.

Our construction can be extended to achieve efficient witness update while still tolerating corrupted accumulator managers although under stronger assumptions (see Section 4). Regarding the efficiency of accumulator updates, we only achieve logarithmic dependency on n. In practice, such dependency may be appropriate for many applications.

3 Our Scheme

We assume that there exist a public broadcast channel with memory. Depending on the level of security required, this can be a simple trusted web server, or a bulletin board that guarantees that every participant can see the published information and that nobody can delete posted message. For a discussion on bulletin boards and an example of their use in another cryptographic protocol, the interested reader is referred to [6]. We rely on broadcast channels in order to ensure that the publication of the successive

accumulator values that correspond to updates of the set cannot be forged. In particular, an adversary who controls the manager of the accumulator cannot publish different accumulator values to different groups of participants.

3.1 Preliminaries

Our scheme is inspired by time stamping systems like those described in [3,2]. In these systems a document needs to be associated to a certain moment in time. The solution proposed there is to divide the time in periods (e.g. hours, days), and place each document as a leaf at the bottom of a binary tree (say, T) with other documents that belong to the same period of time, say t. Then the values associated to each pair of leaves with the same parent node are hashed in order to derive the value of the parent node. This process is repeated until the value v of the root node of the tree is computed. This value v is then published as a representative of the tree T for period t. Later, a given document m can be proven to belong to a certain period of time t by presenting a valid subtree of tree T corresponding to time period t that includes the document m.

We use the above approach to build an accumulator scheme that works for dynamic sets and also allows proofs of nonmembership. In this case, building a proof of nonmembership is somehow similar to the trick of Kocher (in [10]) — instead of storing elements of the set, we store pairs of consecutive elements of the set. Then, proving that an element x is *not* in the accumulated set X amounts to simply proving that there exists elements x_α and x_β, $x_\alpha < x < x_\beta$, such that a pair (x_α, x_β) is stored in the tree.

Our solution uses collision-resistant hash functions, which we formalize as families of functions. In practice we can use a well known hash function like SHA-256, for example. We start recalling the standard notion of collision-resistant hash functions.

Definition 4. *A hash-function family is a function $\mathcal{H} : K \times M \to Y$ where K and Y are non-empty sets and M and Y are sets of strings.*

Definition 5. *(Collision-Resistance) Let $\mathcal{H} : K \times M \to Y$ be a hash-function family. Let k be a security parameter, where $k = |K| = |Y|$. Then $\mathcal{H}$ is collision-resistant if and only if for every polynomial time probabilistic algorithm A we have:*

$$Pr[\kappa \xleftarrow{R} K; (m, m') \leftarrow A(k) : m \neq m',\ \mathcal{H}_\kappa(m) = \mathcal{H}_\kappa(m')] = neg(k)$$

where $\kappa \xleftarrow{R} K$ means that κ is selected uniformly at random in the set of keys K.

In the following H will denote a randomly selected function of a collision-resistant hash-family function $\mathcal{H} : K \times M \to Y$, where M is the set of all binary strings and Y is the set $\{0,1\}^k$, for a large enough security parameter $k \in \mathbb{N}$.

We assume the set X we want to accumulate is ordered and denote by x_i the i^{th} element of $X = \{x_1, x_2, ..., x_n\}$, $n \in \mathbb{N}$. Let $x_0 = -\infty$ and $x_{n+1} = +\infty$ two special elements such that $-\infty \prec x_j \prec +\infty$ for all $x_j \in X$, where $\prec$ is the order relation on X (for example, the lexicographic order on bit strings).

Observe that showing $x \in X$ is equivalent to proving that:

$$(x_\alpha, x_\beta) \in \{(x_i, x_{i+1}) : 0 \leq i \leq n\} \quad \wedge \quad (x = x_\alpha \vee x = x_\beta).$$

On the other hand, showing that $x \notin X$ corresponds to proving:

$$x_\alpha \prec x \prec x_\beta \quad \wedge \quad (x_\alpha, x_\beta) \in \{(x_i, x_{i+1}) : 0 \leq i \leq n\}.$$

Consider now the following recursive definition of *labeled binary tree* T:

- T equals the empty tree Nil, or
- $T = (S; left, right)$ where S is a label (string) and $Left(T) = left$ and $Right(T) = right$ are trees.

Here *left* and *right* are the *left* and *right* child of T respectively. Each tree T has associated a node $N = node(T)$ which is called the root of T as well as the *parent* of $Left(T)$ and $Right(T)$. Each node $N = node(S; left, right)$ has associated a string $Label(N) = S$. Sometimes we identify the tree with its root and we write $Label(T)$ to denote $Label(node(T))$. We say that N' is a node of T if $N' = node(T)$ or N' is a node of $Left(T)$ or $Right(T)$. A *leaf* is a node of the form $(S; Nil, Nil)$. If $T = Nil$, then we say that T has depth 0 and denote it as $depth(T) = 0$. Otherwise, let $depth(T) = 1 + \max\{depth(Left(T)), depth(Right(T))\}$. A tree T is *balanced* if $|depth(Left(T)) - depth(Right(T))| \leq 1$. It is a well known fact that a balanced tree with n nodes has maximum depth $O(\log(n))$.

The set $\{H(x_i||x_{i+1}) : 0 \leq i \leq n\}$ will be called the *base* of X under H. Since H is a collision-resistant hash function and no two x_i are identical, $H(x_i||x_{i+1}) \neq H(x_j||x_{j+1})$ for $i \neq j$, except with negligible probability.

A balanced binary tree T is called a *model* of X under H if:

- For every node N in T there are strings Val_N and $Proof_N$, called node value and node proof respectively, such that $Label(N) = (Val_N; Proof_N)$.
- The base of X is $\{Val_N : N \text{ is a node of } T\}$.
- T has $n+1$ nodes.
- $Proof_N = H(Val_N||Proof_{Left(N)}||Proof_{Right(N)})$ for every node N of T (where $Proof_{Nil}$ corresponds to the empty string).

Figure 1 depicts a toy example of a model of a set.

A subtree T' of a labeled binary tree T is a tree such that: (a) $Label(T') = Label(T)$, (b) $Left(T')$ is a subtree of $Left(T)$ or $Left(T') = Nil$, and (c) $Right(T')$ is a subtree of $Right(T)$ or $Right(T') = Nil$.

Let T be a labeled binary tree. We denote its collection of node values by $\mathcal{V}(T)$. We say that $\mathcal{V} \subseteq \mathcal{V}(T)$ *generates a minimal subtree* U *of* T if U is a subtree of T obtained by taking all nodes in T that belong to all paths from T's root to a node whose value is in $\mathcal{V}$ (the paths include both the root of T and the nodes of value in $\mathcal{V}$) and all the children of these nodes. Figure 2 illustrates the concept of minimal subtree. If U is generated by a singleton $\{S\}$, then we say that U is generated by S.

Proposition 2. *Let $\mathcal{H}: K \times M \to Y$ be a collision-resistant hash function family and H a uniformly chosen function in $\mathcal{H}$. Let $X \subset M$ be an adversarially-chosen polynomial size set (on the security parameter k), and T be a model of X under H. Then, given T, no adversary can efficiently compute a labeled binary tree T' and a value V such that $V \in \mathcal{V}(T') \setminus \mathcal{V}(T)$ and $Proof_{T'} = Proof_T$, except with negligible probability.*

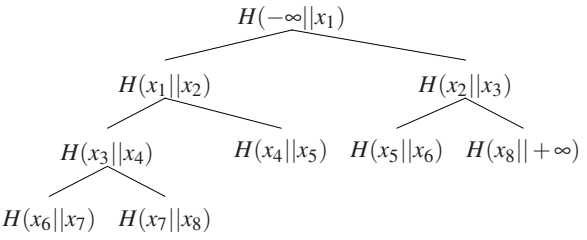


Fig. 1. A tree model T of the set $X = \{x_1, \ldots, x_8\}$. Only node values are shown. Note that the place of the values in the tree is irrelevant.

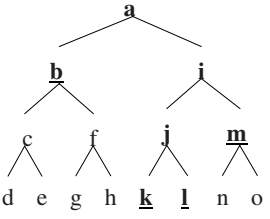


Fig. 2. A tree and its minimal subtree (nodes with values in boldface) generated by the node of value j. Children of the nodes that are on the path from j to a are underlined.

Proof. Let A be a polynomial time stateful adversary which works in two phases. First, on input the security parameter and a hash function $H \in \mathcal{H}$, A outputs a set $X \subset M$ of size polynomial on k. Then, given a model T for X under H, it outputs a labeled binary tree T' and a value V satisfying the conditions of the proposition. Since $Proof_{T'} = Proof_T$ and value V is in $\mathcal{V}(T')$ but not in $\mathcal{V}(T)$ there must exist a node N' in T' and a node N in T such that $Proof_{N'} = H(Val_{N'}||Proof_{Left(N')}||Proof_{Right(N')})$ and $Proof_N = H(Val_N||Proof_{Left(N)}||Proof_{Right(N)})$ are equal but $Val_N||Proof_{Left(N)}||Proof_{Right(N)} \neq Val_{N'}||Proof_{Left(N')}||Proof_{Right(N')}$. Nodes N and N' can be found efficiently by simply traversing both trees in some fixed order.

Now, let B be an adversary that is given a uniformly selected at random collision-resistant hash function $H \in \mathcal{H}$. B first queries A to obtain a set X which it uses to build a model T for X under H. Then, B runs A as a subroutine to obtain another labeled binary tree T' and a value V such that $Proof_T = Proof_{T'}$ and $V \in \mathcal{V}(T') \setminus \mathcal{V}(T)$. Finally, following the procedure mentioned above, B will be able to find a collision for H.

3.2 A Strong Universal Accumulator with Memory Using Hash Trees

In this section we use hash trees to build a universal accumulator with memory.

THE CONSTRUCTION. Let $k \in \mathbb{N}$ be the security parameter and let $X = \{x_1, x_2, \ldots, x_n\}$ be a subset of $M = \{0,1\}^k$. We define the accumulator scheme HashAcc below.

- The memory $\mathfrak{m}$ is a model of X.
- Setup: The algorithm first sets X equal to the empty set. Then, it picks a hash function H uniformly at random from the family $\mathcal{H}$ by first computing a random index $i \in K$ (say using standard multiparty computation techniques among all participants, including the accumulator manager) and then setting $H = H_i$.[1] The algorithm then initializes $\mathfrak{m}$ to a single root node $N_{\mathfrak{m}}$ with value $H(-\infty||+\infty)$. Finally, the accumulator manager publishes $\mathfrak{Acc}_{init} = Proof_{N_{\mathfrak{m}}}$.
- Witness: On input $x \in M$ and memory $\mathfrak{m}$, it computes the witness $w = (w_1, w_2)$ as follows. First, the algorithm sets $w_1 = (x_\alpha, x_\beta)$ where $x = x_\alpha$ or $x = x_\beta$ if $x \in X$. Otherwise, if $x \notin X$ the algorithm sets $w_1 = (x_\alpha, x_\beta)$ where $x_\alpha \prec x \prec x_\beta$. Finally, it sets w_2 as the minimal subtree of $\mathfrak{m}$ generated by the value $H(x_\alpha||x_\beta)$.
- Belongs: On input $x \in M$, witness $w = ((x', x''), u)$, and accumulator value $\mathfrak{Acc}$, it first checks if the following conditions hold: (a) $Proof_u = \mathfrak{Acc}$, (b) $H(x'||x'') \in \mathcal{V}_u$, (c) ($x = x'$ or $x = x''$), and (c') ($x' \prec x \prec x''$). The algorithm outputs 1 if conditions (a), (b), and (c) hold; it outputs 0 if (a), (b), and (c') hold. Otherwise, it outputs $\perp$.
- $\mathsf{Update}_{\mathsf{op}}$: On input an element $x \in M$, an accumulator value $\mathfrak{Acc}_{before}$, and a memory $\mathfrak{m}_{before}$, it proceeds as follows. Consider two cases depending on whether the update is an addition ($\mathtt{op} = \mathtt{add}$) or a deletion ($\mathtt{op} = \mathtt{del}$).

  If $\mathtt{op} = \mathtt{add}$ and $x \notin X$, the algorithm adds x into X by modifying $\mathfrak{m}_{before}$ as follows:

  1. It replaces the value $H(x_\alpha||x_\beta)$ from the appropriate node in $\mathfrak{m}_{before}$ (where $x_\alpha \prec x \prec x_\beta$) by the value $H(x_\alpha||x)$.
  2. It augments the tree $\mathfrak{m}_{before}$ with a new leaf N of value $H(x||x_\beta)$ so the resulting tree $\mathfrak{m}_{after}$ is a balanced tree. Let $V_{Par(N)}$ be the (parent) node where N is attached as a leaf.

  The resulting tree is denoted $\mathfrak{m}_{after}$. Figure 3 illustrates the process of inserting an element into $\mathfrak{m}_{before}$.

  Once tree $\mathfrak{m}_{after}$ is built, the new accumulator is simply the value of the root of the tree, namely $\mathfrak{Acc}_{after} = Proof_{\mathfrak{m}_{after}}$. The witness $w_{add} = (\mathtt{add}, w_{add,1}, w_{add,2})$ that the update (addition) has been done correctly is computed as follows:

  - $w_{add,1}$ corresponds to the minimal subtree of $\mathfrak{m}_{before}$ generated by the set $\{H(x_\alpha||x_\beta), Val_{V_{Par(N)}}\}$, and,
  - $w_{add,2}$ corresponds to the minimal subtree of $\mathfrak{m}_{after}$ generated by the set $\{H(x_\alpha||x), H(x||x_\beta)\}$.

  If $\mathtt{op} = \mathtt{del}$, deleting x from X is done in a similar way as follows. First, the update algorithm locates the two nodes of $\mathfrak{m}_{before}$ that contain x. Let V_α and V_β be those nodes, and let $H(x_\alpha||x)$ and $H(x||x_\beta)$ be their respective values, for some $x_\alpha \prec x \prec x_\beta$. The goal is to remove these nodes and replace them with a new node with value $H(x_\alpha||x_\beta)$ in a way that the derived tree is still balanced. This is done by first replacing V_α with the single node with value $H(x_\alpha||x_\beta)$, and then replacing V_β with a leaf node L (for example, the rightmost leaf on

[1] A common heuristic to avoid interaction is to simply pick H =SHA-256 [12], for example.

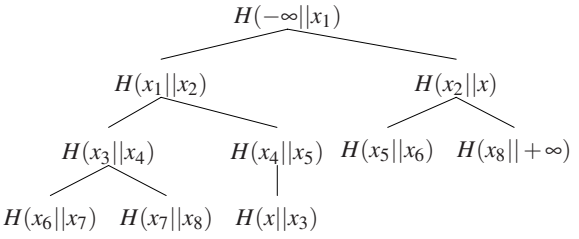


Fig. 3. Inserting x into the tree of Figure 1 where $x_2 \prec x \prec x_3$

the last level of the tree). These replacements yield a new tree $\mathfrak{m}_{after}$ whose root label is set to the value of the accumulator $\mathfrak{Acc}_{after} = Proof_{\mathfrak{m}_{after}}$. The witness $w_{del} = (\mathtt{del}, w_{del,1}, w_{del,2}, w_{del,3})$ is then computed as follows:

- $w_{del,1}$ corresponds to the minimal subtree of $\mathfrak{m}_{before}$ generated by the set $\{H(x_\alpha||x), H(x||x_\beta), Val_L\}$,
- $w_{del,2}$ is the pair $(x_\alpha||x_\beta)$ such that $x_\alpha \prec x \prec x_\beta$, and
- $w_{del,3}$ is the minimal subtree of $\mathfrak{m}_{after}$ generated by $H(x_\alpha||x_\beta)$.

The algorithm $\mathsf{Update}_{\mathtt{op}}$ outputs the new accumulator value $\mathfrak{Acc}_{after}$, the modified memory $\mathfrak{m}_{after}$, and the update witness $w_{\mathtt{op}}$.

- CheckUpdate: On input an element $x \in M$, two accumulator values $\mathfrak{Acc}_{before}$, $\mathfrak{Acc}_{after}$, and an update witness w, it proceeds as follows. If $w = (\mathtt{add}, w_1, w_2)$ then, the algorithm outputs 1 provided that:
  - w_1 is a tree obtained by adding a leaf to w_2,
  - Except for the node of value $H(x_\alpha||x_\beta)$ (for $x_\alpha \prec x \prec x_\beta$) all nodes which are common to w_1 and w_2 have the same value in either one of the trees,
  - $Proof_{w_1} = \mathfrak{Acc}_{before}$ and $Proof_{w_2} = \mathfrak{Acc}_{after}$, and
  - $H(x_\alpha||x), H(x||x_\beta) \in \mathcal{V}(w_2)$.

  Otherwise, it outputs 0. We omit the case $w = (\mathtt{del}, w_1, w_2, w_3)$ which is similar.

SECURITY. We now prove that the scheme HashAcc of the previous section is secure under Definition 3.

First, note that if memory $\mathfrak{m}$ is a model of X, then the memory obtained after executing Update in order to add a new element $x \notin X$, is a model of $X \cup x$. Indeed, suppose $x_\alpha \prec x \prec x_\beta$ and let $H(x_\alpha||x_\beta)$ be the value of a node V in $\mathfrak{m}$. By replacing node V with the node of value $H(x_\alpha||x)$ and adding the node of value $H(x||x_\beta)$, we clearly obtain a set of values $\{H(x_i||x_{i+1}), 0 \le i \le n+1\}$ that corresponds to the successive intervals of the set $X \cup \{x\}$ (where $n = |X|$).

Intuitively, CheckUpdate must guarantee that the updated memory (tree) used to compute the new accumulated value still has the property of having all the successive intervals of the accumulated set as node values, that each interval appears once and only once in the tree, and that no other node value can belong to the tree.

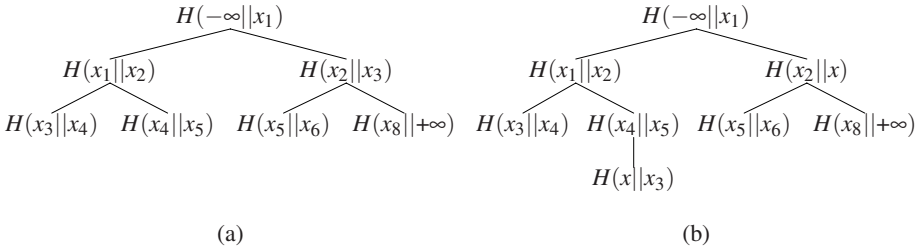


Fig. 4. (a) The minimal subtree of the tree shown in figure 1 and generated by $\{H(x_2||x_3), H(x_4,x_5)\}$. (b) The minimal subtree of the tree shown in Figure 3 and generated by $\{H(x_2||x), H(x||x_3)\}$.

Theorem 1. *Let $\mathcal{H}: K \times M \to Y$ be a collision-resistant hash function family. Then, the accumulator scheme* HashAcc *is a secure strong universal accumulator scheme (with memory).*

Proof. We need to prove the properties *Consistency*, *Addition*, and *Deletion*.

- *(Consistency)* First, we note that $\mathfrak{Acc} \Rightarrow X$ implies that there exists a memory $\mathfrak{m}$ which is a model of X. Let us now suppose that there is an adversary A that can compute a value x and two witnesses w_1, w_2 such that $\mathsf{Belongs}(x, w_1, \mathfrak{Acc}) = 1$ and $\mathsf{Belongs}(x, w_2, \mathfrak{Acc}) = 0$. We assume without lost of generality that $x \in X$. Any such adversary A is in fact able to find x_α and x_β, $x_\alpha \prec x \prec x_\beta$, such that $H(x_\alpha||x_\beta)$ belongs to $\mathcal{V}(\mathfrak{m})$. Since $\mathfrak{m}$ is a model for X, by Proposition 2 this adversary will only succeed with negligible probability. The argument for $x \notin X$ is analogous.
- *(Secure Addition)* Consider the case where the update is the addition of a value x such that $x_\alpha \prec x \prec x_\beta$ and $H(x_\alpha, x_\beta)$ belongs to the base of X, where $\mathfrak{Acc}_{before} \Rightarrow X$. Assume that $\mathsf{CheckUpdate}(x, \mathfrak{Acc}_{before}, \mathfrak{Acc}_{after}, w) = 1$ where both x and $w = (\mathsf{add}, U_{before}, U_{after})$ are arbitrarily chosen by the adversary, and $\mathfrak{Acc}_{after} \not\Rightarrow X \cup \{x\}$. Then, for some two elements $u, v \in M$ the adversary is effectively able to build a tree $S^* = U_{after}$ containing a value $H(u||v)$ that does not belong to $(\mathcal{V}(\mathfrak{m}_{before}) \cup \{H(x_\alpha||x), H(x||x_\beta)\}) \backslash \{H(x_\alpha||x_\beta)\} = \mathcal{V}(\mathfrak{m}_{after})$ and such that in addition $Proof_{S^*} = Proof_{U_{after}} = \mathfrak{Acc}_{after} = Proof_{\mathfrak{m}_{after}}$. This contradicts Proposition 2.
- *(Secure Deletion)* This case is similar to the addition of an element.

EFFICIENCY. We analyze the computational efficiency of the proposed scheme.

Theorem 2. *Let n be the size of X. The witnesses of (non)membership and of updates have size $O(\log(n))$. The update process* Update*, the verification processes* Belongs *and* CheckUpdate *can be done in time $O(\log(n))$.*

Proof. It is enough to show that a minimal subtree U of T generated by a constant number of node values has a size $O(\log(n))$. Indeed, first note that a minimal subtree of a tree generated by a constant number of node values is the union of the minimal subtrees generated by each of the values. It is easy to see that the size of a minimal

subtree generated by a node value is proportional to the depth of the node. This, and the fact that T is balanced, implies the desired conclusion.

4 Other Considerations and Further Extensions

EFFICIENCY. Our solution is theorically less efficient than the scheme proposed in [11]. Nonetheless, if one considers practical instances of these schemes the difference effectively vanishes as in most implementations hash functions operations are significantly faster than RSA exponentiations – which is the core operation used by the schemes in [11,5]. Table 2 shows the time taken by one single RSA exponentiation versus the time taken by our scheme for update operations as a function of the number of the accumulated elements. For the time measurements, we used the *openssl* benchmarking command (see [13]) on a personal computer. Notice that RSA timings were obtained using signing operations, as in the scheme proposed in [11] where exponents may not be small. Timings for SHA operations were measured using an input block of 1024 bits. The comparison is based on the fact that our scheme requires at most $4 \times 2\log(N)$ hash computations, where N is the number of accumulated elements, given that at most four branches of the Merkle tree used in our construction (three for $w_{del,1}$ and one for $w_{del,3}$, see Section 3.2) will have to be recomputed in the case of deletions.

Our results show that even for large values of N using a hash-based scheme is still very efficient. Moreover, our scheme is faster than using a single RSA operation with a 2048-bit key.

DYNAMIC ACCUMULATORS AND WITNESS UPDATE. Our construction as described in Section 3.2 does not allow to update witnesses ([5]). This feature allows a user to recompute her witness after a new value is added or removed from the set X of accumulated values so the new witness can be verified against the new accumulator value. A

Table 1. Running time for RSA and SHA operations

Algorithm	Note	Operations per second
SHA-256	input block of 1024 bits	65507
SHA-512	input block of 1024 bits	16856
RSA-512	signing operation	1179
RSA-1024	signing operation	236
RSA-2048	signing operation	40

Table 2. Comparison of performance between simple RSA exponentiation and logarithmic number of computations of SHA. N is the number of elements that are accumulated. Time is represented in milliseconds.

N	RSA-512	RSA-1024	RSA-2048	SHA-256	SHA-512
2^3	0,845	4,23	25	0,37	1,42
2^{10}	0,845	4,23	25	1,22	4,75
2^{20}	0,845	4,23	25	2,44	9,5
2^{30}	0,845	4,23	25	3,66	14,24

simple way to achieve this is to keep track of each *accumulator value* $\mathfrak{Acc}$ using the construction of Camenisch and Lysyanskaya [5]. Concretely, let $\mathfrak{Acc}_{CL}$ be the accumulator scheme proposed in [5]. Under our modified scheme, a witness that $x \in X$ is now composed by two parts: a witness as computed by our construction and a witness that $\mathfrak{Acc}$ has been accumulated into the accumulator $\mathfrak{Acc}_{CL}$. Since the construction in [5] allows both public updates to the accumulator as well as public computation of membership witnesses, we can update the two-part witness now by simply updating the witness that $\mathfrak{Acc}$ was at some moment a value for the accumulator in our original scheme. Notice that this combined scheme allows the efficient update of witnesses while still preserving the main security property of our scheme (security against a malicious accumulator manager). Obviously, the new feature comes at a cost: lower efficiency and new security assumptions (strong RSA). How to avoid these costs is an open problem.

5 The e-Invoice Factoring Problem

In this section we describe an application of strong universal accumulators that yields an electronic analog of a mechanism called *factoring* through which a company, henceforth referred to as the Provider (P), sells a right to collect future payment from a company Client (C). The ensuing discussion is particularly concerned with the transfer of payment rights associated to the turn over of invoices, that is, *invoice factoring*. The way invoice factoring is usually performed in a country like Chile is that P turns a purchase order from C to a third party, henceforth referred to as Factoring Entity (FE). The latter gives P a cash advance equal to the amount of C's purchase order minus a fee. Later, FE collects payment from C.

There are several benefits to all the parties involved in a factoring operation. The provider obtains liquidity and avoids paying interests on credits that he/she would otherwise need (it is a common practice for some clients as well as several trading sectors in Chile to pay up to 6 months after purchase). The client gets a credit at no cost and is able to perform a purchase for which he might not have found a willing provider.

The main phases of a factoring operation are summarized below: (a) C requests from P either goods or services, (b) P delivers the goods/services to C, (c) P makes a factoring request to FE, (d) FE either rejects or accepts P's request — in the latter case FE gives P a cash advance on C's purchase, (e) later, FE asks C to settle the outstanding payment, and finally (f) C pays FE.

A risk for FE is that P can generate fake invoices and obtain cash advances over them. This danger is somewhat diminished by the fact that such dishonest behavior has serious legal consequences. More worrisome for FE is that P may duplicate real invoices and request cash advances from several FEs simultaneously. But, Chile's local practice makes this behavior hard to carry forth. Indeed, invoices are printed in blocks, serially numbered and pressure sealed by the local IRS agency (known as *Servicio de Impuestos Internos (SII)*). A FE will request the physical original copy of an invoice when advancing cash to P. It is illegal, and severely punished, to make fake copies or issue unsealed invoices.

Approximately half a decade ago, an electronic invoicing system began operating in Chile. Background and technical information concerning this initiative can be downloaded from the website of the SII, specifically from [8].

The newly deployed electronic invoicing system has been widely successful. It has been hailed as a major step in the government modernization. Furthermore, it has created strong incentives for medium to small size companies to enter the so called "information age". Nevertheless, the system somewhat disrupts the local practice concerning factoring. Specifically, a FE will not be able to request the original copy of an invoice, since in a digital world, there is no difference between an original and a copy. This creates the possibility of short term large scale fraud being committed by unscrupulous providers. Indeed, a provider can "sell" the same invoice to many distinct FEs. We refer to the aforementioned situation created by the introduction of electronic invoicing as the *e-Invoice Factoring Problem*. In the full version of this paper we show how to address this problem using strong universal accumulator schemes.

6 Conclusion

We introduced the notion of strong universal accumulator scheme, which provide almost the same functionality as do the universal accumulator schemes defined in [11], namely (a) a set is represented by a short value called accumulator, (b) it is possible to add and remove elements dynamically from the (accumulated) set, and (c) proofs of membership and nonmembership can be generated using a witness and the accumulated value. In this notion, however, the accumulator manager does not need to be trustworthy and might be compromised by an adversary.

We also give a construction of a strong universal accumulator scheme based on cryptographic hash functions which relies on a public data structure to compute accumulated values and witnesses (of membership and nonmembership in the accumulated set). We argue that the proposed scheme is practical and efficient for most applications. In particular, we discuss an application to a concrete and relevant problem — the e-invoice factoring problem.

References

1. Barić, N., Pfitzmann, B.: Collision-free accumulators and fail-stop signed scheme without trees. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 480–494. Springer, Heidelberg (1997)
2. Bayer, D., Haber, S., Stornetta, W.S.: Improving the efficiency and reliability of digital time-stamping. In: Capocelli, R.M., DeSantis, A., Vaccaro, U. (eds.) Sequences II: Methods in Communication, Security, and Computer Science, pp. 329–334. Springer, Heidelberg (1993)
3. Benaloh, J., De Mare, M.: One-way accumulators: A decentralised alternative to digital signatures. In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 274–285. Springer, Heidelberg (1994)
4. Boneh, D., Venkatesan, R.: Breaking RSA not be equivalent to factoring. In: Nyberg, K. (ed.) EUROCRYPT 1998. LNCS, vol. 1403, pp. 59–71. Springer, Heidelberg (1998)
5. Camenisch, J., Lysyanskaya, A.: Dynamic accumulators and application to efficient revocation of anonymous credentials. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 61–76. Springer, Heidelberg (2002)

6. Cramer, R., Gennaro, R., Schoenmakers, B.: A secure and optimally efficient multi-authority election scheme. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 103–118. Springer, Heidelberg (1997)
7. Damgård, I.: Collision free hash functions and public key signature schemes. In: Price, W.L., Chaum, D. (eds.) EUROCRYPT 1987. LNCS, vol. 304, pp. 203–216. Springer, Heidelberg (1988)
8. Servicio de Impuestos Internos. Información sobre factura electrónica [June 19, 2008], https://palena.sii.cl/dte/mn_info.html
9. Fazio, N., Nicolisi, A.: Cryptographic accumulators: Definitions, constructions and applications (2003) [June 19, 2008], http://www.cs.nyu.edu/~nicolosi/papers/accumulators.ps
10. Kocher, P.C.: On certificate revocation and validation. In: Hirschfeld, R. (ed.) FC 1998. LNCS, vol. 1465, pp. 172–177. Springer, Heidelberg (1998)
11. Li, J., Li, N., Xue, R.: Universal accumulators with efficient nonmembership proofs. In: Katz, J., Yung, M. (eds.) ACNS 2007. LNCS, vol. 4521. Springer, Heidelberg (2007)
12. National Institute of Standards and Technology (NIST). FIPS Publication 180: Secure Hash Standard (SHS) (May 1993)
13. OpenSSL Project. OpenSSL Package (June 2008) [June 19, 2008], http://www.openssl.org

A Novel Audio Steganalysis Based on High-Order Statistics of a Distortion Measure with Hausdorff Distance

Yali Liu[1], Ken Chiang[2], Cherita Corbett[2], Rennie Archibald[3], Biswanath Mukherjee[3], and Dipak Ghosal[3]

[1] Electrical & Computer Engineering, University of California, Davis, Davis, CA 95616 USA
yliu@ece.ucdavis.edu
[2] Sandia* National Laboratories, Livermore, CA 94551, USA
[3] Department of Computer Science, University of California, Davis, Davis, CA 95616 USA

Abstract. Steganography can be used to hide information in audio media both for the purposes of digital watermarking and establishing covert communication channels. Digital audio provides a suitable cover for high-throughput steganography as a result of its transient and unpredictable characteristics. Distortion measure plays an important role in audio steganalysis - the analysis and classification method of determining if an audio medium is carrying hidden information. In this paper, we propose a novel distortion metric based on Hausdorff distance. Given an audio object x which could potentially be a stego-audio object, we consider its de-noised version x' as an estimate of the cover-object. We then use Hausdorff distance to measure the distortion from x to x'. The distortion measurement is obtained at various wavelet decomposition levels from which we derive high-order statistics as features for a classifier to determine the presence of hidden information in an audio signal. Extensive experimental results for the Least Significant Bit (LSB) substitution based steganography tool show that the proposed algorithm has a strong discriminatory ability and the performance is significantly superior to existing methods. The proposed approach can be easily applied to other steganography tools and algorithms.

1 Introduction

Steganography is the art and science of hiding a secret message within an innocuous and open carrier medium, such as digital audio, image, and video. To achieve covert communications without raising suspicion, media containing some hidden information (stego-objects) should be indistinguishable from media without any hidden information (cover-objects). The rapid proliferation of Voice over

* Sandia is a mutiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-04AL85000.

T.-C. Wu et al. (Eds.): ISC 2008, LNCS 5222, pp. 487–501, 2008.

Internet Protocol (VoIP) and other Peer-to-Peer (P2P) audio services provide vast opportunities for covert communications [1]. By slightly altering the binary sequence of the audio samples with existing steganography tools, covert communication channels may be relatively easy to establish. Moreover, the inherent redundancy in the audio signal and its transient and unpredictable characteristics imply a high hidden capacity. This is further aided by the fact that the human ear is insensitive to small distortions in the audio signal. All these make audio a good candidate for use as a "cover" for covert communications to hide secret messages.

The countermeasure to steganography is steganalysis. In particular, steganalysis seeks to identify suspected information streams; determine whether or not they have hidden messages encoded into them; and, if possible, recover the hidden information. Steganalysis can also serve as an effective way to judge the security performance of steganography techniques, leading to new steganography methods followed by new and improved steganalysis techniques for their detection. As covert communications greatly increase the possibility of unknown malicious activities from a security standpoint, there is significant demand in steganalysis technique to detect hidden information in open digital media content. In this paper, similar to most of the steganalysis work, we focus on the detection of the presence of hidden content, rather than message recovery or other functions.

In recent years, there has been significant effort in the steganalysis of digital images [2] [3] [4]. Typically, natural images tend to be contiguous and smooth, which leads to high spatial coherence among adjacent pixels. Since the hidden message is usually independent of the cover-image, embedding the hidden message into the cover-image may decrease or even destroy the inherent natural correlation. As a result, most of the image steganalysis algorithms attempt to determine some particular statistical features that can capture this change. Although most image steganalysis algorithms claim that they can be easily extended to other types of media files (e.g., audio), many of the models capture statistical regularities inherent to the spatial composition of images, which are not present in other types of media files such as audio [5]. As a result, deeper research must be conducted on the feature extraction when identifying stego-audio files.

Compared to image steganalysis, audio steganalysis is relatively unexplored. Johnson et al. proposed a universal steganalysis algorithm that exploits the statistical regularities of recorded speech [5]. In [6], audio quality metrics were adopted to capture the distortion introduced by the hidden information. Later in [7], Avcibas proposed an audio steganalysis algorithm using content-independent distortion measures. However, all these audio distortion measurements are seeking ways in which the existing quality metrics can reflect the sensitivity to the presence of hidden messages.

In this paper, we propose an audio steganalysis scheme that measures audio distortion using Hausdorff Distance [8]. Among various distance measures, Hausdorff distance was chosen because of its successful applications in matching given

templates in arbitrary target images [9]. Its strong discriminatory power makes it very helpful in the distortion measurement process. High order statistics derived from this distortion measure can then be used to generate features for a classifier. Unlike previous work in audio steganalysis that used the traditional audio quality metrics, such as signal-to-noise ratio (SNR), Perceptual Audio Quality Measure (PAQM) [6], and other such metrics, the proposed distortion measure is designed specifically to detect modifications to pure audio content as follows.

Given an audio object x which could potentially be a stego-audio object, we consider its de-noised version x' as an estimate of the cover-object. After appropriate segmentation, we apply wavelet decomposition to both x and x' to generate wavelet coefficients [10] at different levels of resolution. Next, Hausdorff distances are used to test the similarities between the wavelet coefficients of the audio signals and their de-noised versions. The statistical moments of these Hausdorff distances are used as the features to train a classifier on the difference between known cover-audio objects and stego-audio objects with different hidden content loadings. Simulations with numerous audio sequences show that our algorithm provides significantly higher classification rates than existing schemes that use standard audio quality metrics or statistical moments without considering audio quality. Moreover, as the proposed scheme makes no assumptions about the embedding technique used, it should be easily applicable to other steganography tools and algorithms.

The paper is organized as follows. Section 2 provides the related work. In Section 3, we briefly introduce the general idea of steganalysis based on audio distortion before presenting the novel steganalysis distortion metric based on Hausdorff distance and the corresponding high-order statistics used as a feature vector. In Section 4, we implement our technique and present our experimental results and performance comparisons. Section 5 concludes the paper.

2 Related Work

In recent years, there has been significant research effort in steganalysis with primary focus on digital images. Since there are similarities between images and audios, in this section, we review some image steganalysis algorithms which may be helpful for the audio steganalysis. It should be noted that many steganalysis techniques are specific to some particular data hiding methods [2] [11] [12] [13] [14]. However, since the data-embedding method is typically unknown prior to detection, we focus on the design of a unified steganalysis algorithm to detect the presence of steganography independent of the steganography algorithms used. Moreover, we focus on passive detection as opposed to active warden steganalysis [6] which aim to detect and modify the hidden content.

2.1 High-Order Statistics and Steganalysis

A number of prior studies have shown that high-order statistics are very effective in differentiating stego-images from cover-images. In [15], Farid proposed a general steganalysis algorithm based on image high-order statistics. In this method,

a statistical model based on the first (mean) and higher-order (variance, skewness, and kurtosis) magnitude statistics, extracted from wavelet decomposition, is used for image steganography detection. In [16], a steganalysis method based on the moments of the histogram characteristic function was proposed. It has been proved that, after a message is embedded into an image, the mass center (the first moment) of histogram characteristic function will decrease. In [10], Holotyak et al. used higher-order moments of the probability density function (PDF) of the estimated stego-object in the finest wavelet level to construct the feature vectors. Due to the limited number of features used in the steganalysis technique proposed in [16], Shi et al. proposed the use of statistical moments of the characteristic functions of the wavelet sub-bands [17]. Because the n^{th} statistical moment of a wavelet characteristic function is related to the n^{th} derivative of the corresponding wavelet histogram, the constructed 39-dimensional feature vector has proved to be sensitive to embedded data.

Usually, the steganalysis algorithms based on the high-order statistics can achieve satisfactory performance on image files, regardless of the underlying embedding algorithm. However, these statistical models may not be appropriate for audio files because these models capture statistical regularities inherent to the spatial composition of images which is not present in audio [5].

2.2 Distortion Measures and Steganalysis

The concept of using distortion measures to classify cover-objects and stego-objects was introduced by Avcibas et al. in 2003 [18]. Since the presence of steganography communication in a signal can be modeled as additive noise in the time or frequency domains [16], the de-noised versions of the image signals can be used to represent close approximations of the cover-images. It has been shown that the distortion (measured by the distance in the feature space) of the cover-image to its de-noised version is different than the distortion between a stego-image and its de-noised version. Specifically, some image quality metrics, e.g., Minkowsky [18], correlation, and human visual system (HVS) based measures [19] [20], are selected as the feature set to distinguish between cover-images and stego-images. This concept was extended to audio steganalysis in [6]. Similar to [18], the potential of distortion audio metrics is used to build a steganalyzer to discriminate between cover-audio objects and stego-audio objects. Particularly, the traditional audio quality metrics, such as SNR, PAQM, and other such metrics are tested for their sensitivity to the presence of steganographic content. In [7], Avcibas proposed an audio steganalysis algorithm using content-independent distortion measures. By removing content dependency during the distortion measurement, the paper shows that the discriminatory power is enhanced.

Note that all these algorithms attempt to find good features from the standard quality metrics which are designed to evaluate the perceptual and objective quality performance of images or audio. As the primary motivation for developing these quality metrics was for purposes other than steganalysis, the capability of distinguishing changes in quality due to embedding content using steganography

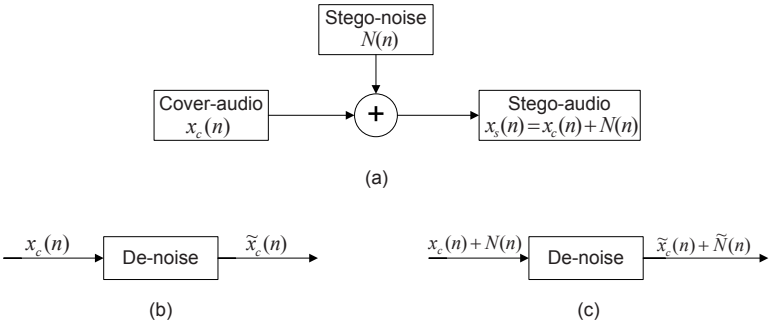


Fig. 1. Schematic descriptions of (a) additive noise steganography model, (b) de-noising a cover-audio object, and (c) de-noising a stego-audio object

may be limited. Consequently, we argue that it is better to define a distortion metric that is designed specifically to detect modifications to audio content. Furthermore, since the high-order moments have been helpful in image steganalysis, we believe they can also contribute in audio steganalysis if used properly.

3 Methodology

In this section, we first review steganography message embedding techniques and set up a steganalyer based on audio distortion. Then based on the approximate additive noise model, an audio steganalysis using high-order statistics of a distortion measure with Hausdorff distance is proposed.

3.1 Steganalysis Based on Audio Distortion

Due to the natural noise in the media transmission process, e.g., quantization, sensor, and channel, a number of steganography hiding schemes try to disguise the hidden message as a naturally present noise. As such, a generalized additive noise scheme has been developed in [20] that is capable of embedding data with any given distribution. Moreover, the work in [15] shows that most of the steganography algorithms, e.g., Least Significant Bit (LSB) steganography, spread spectrum image steganography, or even more robust and stealthy steganography schemes such as Discrete Cosine Transform (DCT) steganography, can be approximated as an additive noise scheme.

The same additive noise model can also be applied to audio files. The steganography message embedding process is shown in Figure 1. Let $x_c(n)$ denote a cover-audio object and $x_s(n)$ be its stego-version. Let $N(n)$ be an independent and identically distributed (i.i.d) Gaussian noise; then the stego-audio object can be expressed as $x_s(n) = x_c(n)+N(n)$ with the additive noise model. A good feature should enlarge the distance between $x_c(n)$ and $x_s(n)$. However, it is important to note that, in a real communication environment, a reference audio file needs

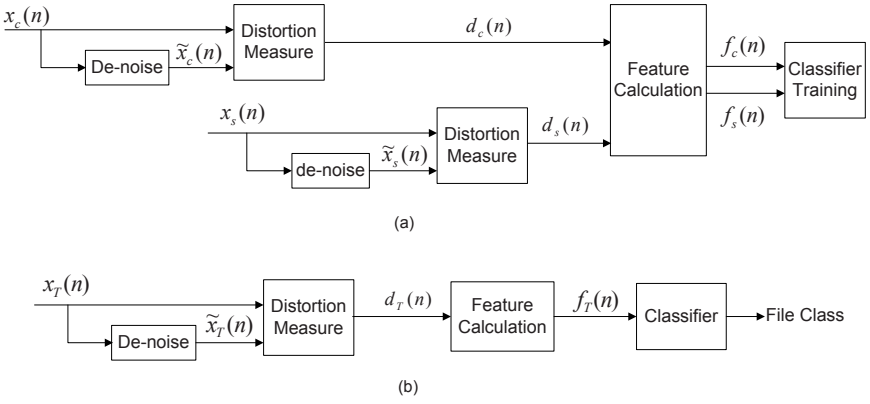


Fig. 2. Schematic description of (a) training and (b) testing for the steganalysis

to be used since we cannot get specific information about the original cover-audio object. The de-noised version of an audio file has already been shown to be a good estimation of the cover signal [6]. Note that the de-noised version of stego-audio is still the de-noised cover-audio with some i.i.d. Gaussian noise.

The training and testing procedures for the steganalysis are shown in Figure 2. Let $\tilde{x_c}(n)$ and $\tilde{x_s}(n)$ be the de-noised versions of a cover-audio object and a stego-audio object, respectively. The defined distortion metric, in fact, is simply trained to differentiate between the distances, denoted as $d_c(n)$ and $d_s(n)$, of the cover-audio object and stego-audio object to their de-noised versions. Instead of using $d_c(n)$ and $d_s(n)$ as audio features, further feature calculation procedures are performed before going to the classifier training process. The test audio file $x_T(n)$ will go through the same procedures of distortion measure and feature selection until the resulting feature vector $f_T(n)$ is achieved, and then used to judge the test file type with the training model.

In addition to feature calculation, the classifier plays an important role in the steganalysis process. It affects the classification performance in terms of success classification rate as well as the computational complexity. In our work, we use the freely available package Library for Support Vector Machines (LIBSVM) [21], which is powerful software for data classification and is widely used in steganalysis.

3.2 Feature Calculation

Wavelet de-noising. The goal of the de-noising process is to recover the characteristics of the original cover-audio object while also removing as much noise as possible. Considering the non-stationary characteristics of audio signals [5], a smoothing filter may not be very suitable for estimating the cover-object. Among many other techniques, Wiener filtering is a powerful tool for additive noise reduction. In its basic form, Wiener theory assumes that signals are stationary processes. However, this assumption is not realistic for audio signals, whose

characteristics change in time and therefore are considered non-stationary signals. As a result, we consider adopting the wavelet de-noising technique. Using the thresholding technique [22], wavelet approximation allows an adaptive representation of signal discontinuities. Wavelets also provide unconditional basis for a variety of function spaces and thus provide better approximation power than Fourier basis to help recover the characteristics of the cover-audio signal more effectively.

Distortion Measure. Once we get the de-noised version of an audio signal, a distance measurement will be applied to measure the distortion or degradation of the original audio signal. Such a measurement should respond to the presence of a hidden message in an accurate, consistent, and monotonic (with respect to the size of the hidden message) way. It should be noted that, instead of gathering information directly from audio files, signatures of the audio files are generated based on their wavelet coefficients at different levels of resolution and will be used to test the distance of the audio file and its de-noised version. The wavelet transform is chosen for its well-known capability of multi-resolution decomposition [23], which can help to enlarge the influence of the additive noise present as a result of embedding. Since the hidden information may only modify a small portion of the cover-objects, the distortion is calculated at their predefined small segments separately. At this point, the time-frequency localization [23] characteristic of the wavelet transform may also provide some information about the discontinuities that occur.

Since the distortion metric should be sensitive to the presence of a hidden message and its reaction should be proportional to the embedding strength, Hausdorff distance [8] is used as a dissimilarity measure. Among dissimilarity measures over binary images, the Hausdorff distance has often been used in the content-based retrieval domain and is known to have successful applications in object matching [18]. On the other hand, Hausdorff distance is very sensitive to noise [24] [9]. A small distortion can result in a significant distance between two objects. However, in steganalysis, the main issue under consideration is not the content of an audio file but the minor distortions introduced during the data-hiding process. As a result, this characteristic of Hausdorff distance makes it very helpful in the steganalysis.

The Hausdorff distance is basically a max-min distance. Suppose the length of each segment of the audio file is M. After de-noising and wavelet decomposition at level p, for m^{th} segment, the wavelet coefficients of the audio file and its de-noised version are $C_m^p = \{C_m^1, C_m^2, \ldots, C_m^q\}$ and $\tilde{C}_m^p = \{\tilde{C}_m^1, \tilde{C}_m^2, \ldots, \tilde{C}_m^q\}$, where $q = M/2^p$. Then, its distortion measure with Hausdorff distance is:

$$H_m^p = max\{h(C_m^p, \tilde{C}_m^p), h(\tilde{C}_m^p, C_m^p)\} \tag{1}$$

where

$$h(C_m^p, \tilde{C}_m^p) = max_{i=1,2,\ldots,q}\{min_{j=1,2,\ldots,q}\|c_m^j - \tilde{c}_m^j\|\} \tag{2}$$

is the directed Hausdorff distance from C_m^p to $\tilde{C}_m^p$ and $\|c_m^j - \tilde{c}_m^j\|$ is some underlying norm on the point of c_m^i and $\tilde{c}_m^j$. Here, we use the absolute difference.

Feature Calculation. As in [14], to get good local distortion estimation, the segment size M that the audio file is split into should not be very large (in our experiment, we set M as 1024 audio samples). As a result, the number of Hausdorff distances after the distortion measurement is still very large. It is unrealistic to use these distances directly for steganalysis. A feasible approach is to extract a certain amount of data (features) from these distances and use them to represent the distortion measurement for steganalysis. Because the task of the segmentation is to test the distortion regularity for the audio files, high-order statistics based on the moments will be used as the final feature.

Suppose the entire audio length is L samples, then the total number of segments is $\lceil \frac{L}{M} \rceil$. For wavelet decomposition level p, where $p = 0, 1, \ldots, P$, the overall distortion measured using Hausdorff distance is:

$$D^p = \langle H_1^p, H_2^p, \ldots, H_{\lceil \frac{L}{M} \rceil}^p \rangle \tag{3}$$

and the feature vector $V^p = \langle v_1^p, v_2^p, \ldots, v_K^p \rangle$ can be extracted according to the following equation.

$$v_i^p = \frac{\sum_{j=1}^{n} (f_j^p)^i \cdot d_j^p}{\sum_{j=1}^{n} d_j^p}, i = 1, 2, \ldots, K \tag{4}$$

where d_j^p is the amplitude of j^{th} frequency component f_j^p to the distortion distances D^p and K is the total number of moments.

In this way, for each wavelet decomposition level, we have K features. For a total P-level wavelet decomposition plus the level 0 which is the signal itself, we have $(P + 1) \times K$ features which form a high-dimensional feature vector:

$$V = \langle V^0, V^1, \ldots, V^p \rangle \tag{5}$$

for steganalysis.

3.3 Algorithm Summary

In summary, the proposed feature calculation algorithm proceeds along the following steps:

Step 1. For a given audio file $x(n)$, apply wavelet de-noising to get its de-noised version $\tilde{x}(n)$.

Step 2. Partition the signal $x(n)$ and $\tilde{x}(n)$ with pre-defined segment length M. Calculate the wavelet coefficients c_m^p at different levels p for segment m.

Step 3. For each wavelet decomposition level p, calculate the distortion measure H_m^p with Hausdorff distance in Equation (1) for all the segments.

Step 4. Set up the feature vector V^p by calculating the moments of D^p using Equation (3) for each wavelet decomposition level p.

Step 5. Set up the high-dimensional feature V using Equation (5).

4 Experimental Results

To evaluate the performance of the proposed steganalysis algorithm, we randomly picked 994 wav files from the wav surfer database [25]. All these wav files are parts of movies or television programs and have different audio characteristics. These audio files (compressed to MP3 formats) are transformed into standard PCM wav format using Nero Wave Edit before processing. The sampling rate is 44.1 kHz with 16 bits per sample. The audio file lengths vary from one second to 298 seconds. As for the steganography tool, we have used Steghide [26] due to its robustness against a number of different steganalysis tools. For the results presented in this paper, we have set P (the number of wavelet decomposition levels) to 4 and K (the number of high order moments) to 5. This implies that a 25-D feature vector is generated for each audio file.

4.1 Performance Comparison with Other Audio Steganalysis Algorithms

In this section, we compare the performance of the proposed algorithm with three known algorithms referred to in Section 2. Each experiment randomly selects 895 of the 994 original audio as cover-audio objects. For each audio object, we create a corresponding stego-audio object with a specific amount of hidden content (measured with hidden ratio). As a result, 895 stego-audio files and their original 895 cover-audio files are used for training the classifier. Here, the hidden ratio is the percentage of the size of the hidden message to the hidden capacity (the maximum size of the information that can be hidden) which is determined by Steghide. From the remaining 99 audio files, we obtain 99 pairs, each pair consisting of the original audio and the corresponding stego-audio with a specific hidden ratio. These 99 pairs of audio are used for testing. Note that in this section, we only focus on the feature effectiveness and assume that the hidden ratio information is known before testing, i.e., the hidden ratios are the same in the training and testing processes. The performance metric used is the correct classification rate[1] with the average computed from 10 independent experiments.

Of the three different reference algorithms considered for comparison, the first two were selected to test the importance of high-order statistics in audio steganlalysis. Particularly, one algorithm (HOMWC) [15] is directly based on the high-order moments of the wavelet coefficients of the audio signal. The second algorithm (SMCFWS) [17] is based on statistical moments of the characteristic functions of wavelet sub-bands. In order to make a fair comparison of the performance of the algorithms, we have considered the first 5 moments for the first 4 wavelet decomposition levels as in our algorithm as well. The third algorithm (QMGAQM) [6] is based on the quality measurement with general audio quality metrics. Similarly to the study in [6], a 5-D feature vector based on SNR and PAQM and other such metrics is used to train the classifier.

[1] The correct classification rate is the average detection rate to all the original audio objects and stego-audio objects.

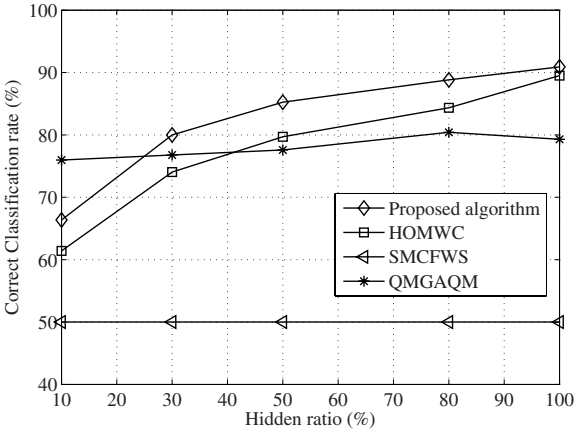


Fig. 3. Comparison of correct classification rate with other audio steganalysis algorithms

Figure 3 plots the correct classification rate as a function of the hidden ratio. It is clearly observed that the correct classification rate achieved by our proposed algorithm is more than 90% with 100% hidden ratio and 85% for 50% hidden ratio. Even with only 10% hidden ratio, our approach can still achieve more than 66% successful detection. More importantly, the proposed algorithm shows strong monotonic characteristics with different hidden ratios. On the other hand, it is observed that the SMCFWS algorithm does not perform well. Although the algorithm using moments of wavelet coefficients (HOMWC) is fairly good, our algorithm can still get more than 10% improvement in the correct classification rate. This does not come as a surprise since these algorithms work very well in the stego-image identification due to their ability to capture the statistical regularities inherent in the spatial composition of images which are not present in audio. Note that the performance of the algorithm with audio quality metrics (QMGAQM) is fairly good at low hidden ratio. However, the classification rate does not show strong monotonic characteristics with respect to different hidden ratios. This confirms our aforementioned doubt that the standard audio quality metrics may or may not reflect modifications to pure audio content.

4.2 Performance with Respect to Different Hidden Ratios

In the previous section, we compared the feature effectiveness with different audio steganalysis algorithms using the same hidden ratio at the training process and testing. However, in a real system, the hidden ratio information will be unknown before the test. In this section, we evaluate the performance of the proposed algorithm by considering different hidden ratios during the training process. Similar to the previous section, we randomly select 895 original audio files as cover-audio objects. Their corresponding stego-audio files are generated based on five different hidden ratios: 10%, 30%, 50%, 80%, and 100%. For each

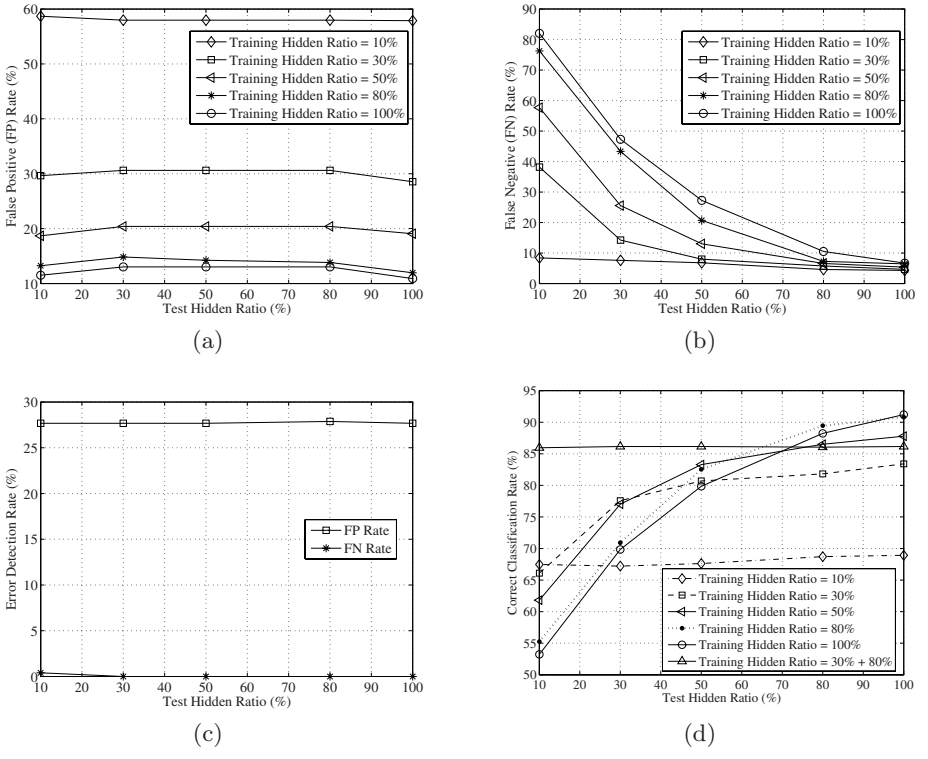


Fig. 4. Detection performance at different hidden ratios. (a) FP rate with different training hidden ratios; (b) FN rate with different training hidden ratios; (c) FP rate and FN rate with the $30\% + 80\%$ training hidden ratio; (d) correct classification rate with different training hidden ratios.

hidden ratio, we created 895 stego-audio objects and used them in conjunction with the original 895 cover-audio objects for training the classifier. This leaves 99 pairs of audio files (original audio objects and their stego-versions with the same hidden ratio) to be used for testing. The performance metrics used are the false positive (FP) and false negative (FN) rates reported as an average of ten independent experiments.

Figure 4(a) and 4(b) plot the detection performance at different hidden ratios during the training and testing processes. They show that both FP and FN of our algorithm are influenced by the hidden ratio in the training process. Specifically, the higher the hidden ratios used in the training process, the lower the FP. This is consistent with the fact that distortion is higher with higher hidden ratios. Thus, at high hidden ratios, the test cover-audio object is less likely to be misjudged as a stego-audio object. However, the large distortion introduced by the high hidden ratios in the training process will make the system more likely to miss-classify stego-audio objects with lower hidden ratio. Consequently, there is a trade-off between FP and FN. Concerning this trade-off, we find it is reasonable to train our

SVM models with audio files embedded with multiple hidden ratios. Therefore, during the training process, for each cover-audio object, multiple versions of the stego-audio objects with the selected set of the hidden ratios are used. Considering the unknown properties of the test audio and computation cost for the training process, only limited combinations of the hidden ratios may be selected. In our study, we find 30% and 80% hidden ratios can help to train the system with a good representation of the cover-audio objects and stego-audio objects simultaneously.

Figure 4(c) plots the simulation results for the test audio objects containing different hidden ratios with a training set that contains stego-audio objects with both 30% and 80% hidden ratios (denoted as 30% + 80%). The low FP rate and FN rate indicate that, in most cases, the system can distinguish the cover-audio objects and stego-audio objects successfully. More importantly, these results show that multiple training hidden ratios greatly improve the robustness of our algorithm. Our improved robustness can be observed as both FP rate and FN rate are almost unchanged with different test hidden ratios, which is very helpful since we usually do not know the hidden ratio of a stego-object in advance.

Figure 4(d) plots the correct classification rates by the influence of different hidden ratios in the training process. Note that the smaller the difference between the test hidden ratio and the training hidden ratio, the better classification performance we achieve. Moreover, with the help of multiple training hidden ratios, the correct classification rate shows strong robust characteristics and for most cases the correct classification rate is much higher than the best one we can get with only one training hidden ratio. Although there is still some small gap between the performance of multiple hidden ratios at higher hidden ratios, e.g., 80% and 100%, the improvement in classification can be achieved by increasing the number of training hidden ratios.

4.3 Analysis of Feature Contributions

To measure the effectiveness of each feature in the 25-D feature vector, we define the relative feature distance as:

$$\Delta = \frac{v_s - v_c}{v_s} \times 100\% \tag{6}$$

where v_c and v_s are the feature vectors obtained from cover-audio objects and stego-audio objects, respectively. Figure 5 plots the relative feature distance for 100 audio files randomly selected from our audio database. The hidden ratio in this section is set to be 100%. The results show that the differences between the features of the cover-object and stego-object are less noticeable with the higher wavelet decomposition level. This is because the embedded information corresponds to high frequency noise. In the wavelet decomposition process, the lower levels correspond to the higher frequency bands and higher levels lead to decreasing frequency bands. As a result, the feature at the lower wavelet levels will better detect changes resulting from noise. Finally, Table 1 shows the contribution of each dimension of the 25-D feature vector to the classification performance by separately applying each dimension as a one-dimensional (1-D) feature for steganalysis. The performance is measured with the correct classification rate for

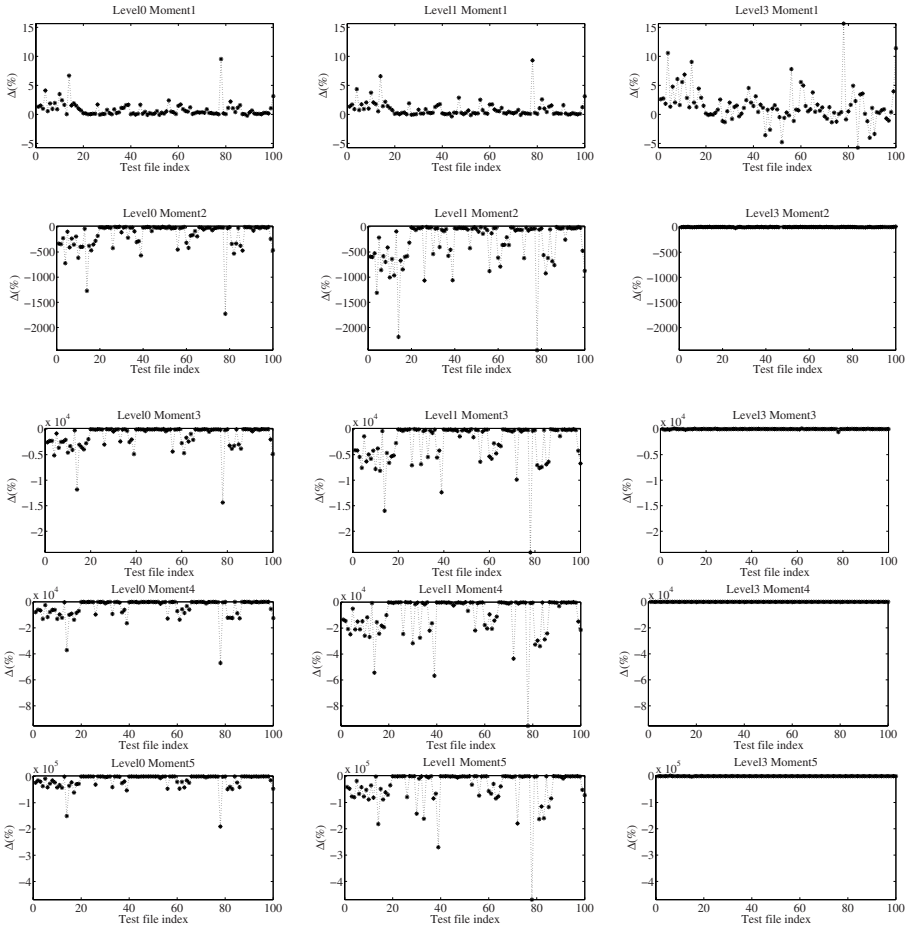


Fig. 5. Feature effectiveness with respect to different wavelet decomposition levels and statistical orders

the randomly-selected 895 cover-audio objects and their stego-versions in the training process. The results show that the correct classification rates are different for different 1-D features. Particularly, the correct classification rate is much higher at the lower wavelet decomposition levels compared to the higher levels, and the original signal gets a median correct classification rate within the different wavelet decomposition levels. Also, within the same level, the correct classification rate increases with the increasing moment order. These results confirm our previous analysis for the different feature effectiveness. In addition, it can be observed that the correct classification rates using any 1-D feature vector or any 5-D feature vector are significantly lower than using the combined 25-D feature vector. In other words, the 25 features collectively perform much better, thus these features are complementary in steganalysis.

Table 1. Correct classification rate of 1-D feature for the training data

Moment Order	Level 0	Level 1	Level 2	Level 3	Level 4
1	55.30	57.42	55.02	51.79	51.06
2	62.94	65.56	55.91	53.01	51.78
3	65.90	77.23	56.19	53.57	52.40
4	66.35	79.80	58.25	55.80	52.57
5	67.18	82.75	59.04	56.25	52.68
5-D feature vector	68.97	85.77	69.87	60.71	54.30
25-D feature vector			89.45		

5 Conclusion

In this paper, we presented an audio steganalysis method that is based on audio distortion measurement and high-order statistics in the feature selection. A distortion metric based on Hausdorff distance was designed specifically to detect modifications and additions to audio media. We considered the de-noised version of the audio object as an estimate of the cover-object. We then used the Hausdorff distance to measure the distortion. The distortion measurement was obtained at various wavelet decomposition levels from which we derived high-order statistics as features for a classifier to determine the presence of hidden information in an audio signal. Results from simulations with numerous audio sequences showed that our algorithm provides significantly higher detection rates than existing schemes that use standard audio quality metrics or statistical moments without considering audio quality.

References

1. Dittmann, J., Hesse, D., Hillert, R.: Steganography and steganalysis in voice-over ip scenarios: operational aspects and first experiences with a new steganalysis tool set. In: Security, Steganography, and Watermarking of Multimedia Contents VII, San Jose, CA, USA, January 2005, pp. 607–618. SPIE (2005)
2. Fridrich, J., Goljan, M., Du, R.: Reliable detection of LSB steganography in color and grayscale images. In: Proceedings of the 2001 workshop on multimedia and security: new challenges, Ottawa, Ontario, Canada, october 2001, pp. 27–30. ACM Press, New York (2001)
3. Johnson, N.F., Jajodia, S.: Steganalysis of images created using current steganography software. In: Proceedings of the Second International Workshop on Information Hiding, Portland, OR, USA, April 1998, pp. 273–289. Springer, Heidelberg (1998)
4. Westfeld, A., Pfitzmann, A.: Attacks on steganographic systems. In: Proceedings of the Third International Workshop on Information Hiding, Dresden, Germany, september 1999, pp. 61–76. Springer, Heidelberg (1999)
5. Johnson, M.K., Lyu, S., Farid, H.: Steganalysis of recorded speech. In: Delp III, E.J., Wong, P.W. (eds.) Security, Steganography, and Watermarking of Multimedia Contents VII, May 2005, vol. 5681, pp. 664–672. SPIE (2005)
6. Ozer, H., Avcibas, I., Sankur, B., Memon, N.D.: Steganalysis of audio based on audio quality metrics. In: Delp III, E.J., Wong, P.W. (eds.) Security and Watermarking of Multimedia Contents V, January 2003, vol. 5020, pp. 55–66. SPIE (2003)

7. Avcibas, I.: Audio steganalysis with content-independent distortion measures. Signal Processing Letters, IEEE 13(2), 92–95 (2006)
8. Huttenlocher, D.P., Klanderman, G.A., Rucklidge, W.J.: Comparing images using the hausdorff distance. Pattern Analysis and Machine Intelligence, IEEE Transactions on 15(9), 850–863 (1993)
9. Veltkamp, R.C.: Shape matching: similarity measures and algorithms. In: Shape Modeling and Applications, SMI 2001 International Conference on, Genova, Italy, pp. 188–197 (May 2001)
10. Holotyak, T., Fridrich, J., Voloshynovskiy, S.: Blind statistical steganalysis of additive steganography using wavelet higher order statistics. In: Lecture Notes in Computer Science, pp. 273–274. Springer, Heidelberg (2005)
11. Chandramouli, R., Memon, N.: Analysis of LSB based image steganography techniques. In: Image Processing, 2001. International Conference on, Thessaloniki, Greece, pp. 1019–1022 (October 2001)
12. Dabeer, O., Sullivan, K., Madhow, U., Chandrasekaran, S., Manjunath, B.S.: Detection of hiding in the least significant bit. Signal Processing, IEEE Transactions on 52(10), 3046–3058 (2004)
13. Dumitrescu, S., Wu, X.: Steganalysis of LSB embedding in multimedia signals. In: Multimedia and Expo, 2002. ICME 2002. IEEE International Conference on, Lusanne, Switzerland, pp. 581–584 (August 2002)
14. Dumitrescu, S., Wu, X., Wang, Z.: Detection of LSB steganography via sample pair analysis. In: Proceedings of the Fifth International Workshop on Information Hiding, Noordwijkerhout, The Netherlands, pp. 355–372 (October 2002)
15. Farid, H.: Detecting hidden messages using higher-order statistical models. In: Image Processing. 2002. International Conference on, Rochester, NY, USA, pp. 905–908 (September 2002)
16. Harmse, J.J.: Steganalysis of additive noise modelable information hiding. Master's thesis, Rensselaer Polytechnic Institute, Troy, New York, USA (2003)
17. Shi, Y.Q., Xuan, G., Yang, C., Gao, J., Zhang, Z., Chai, P., Zou, D., Chen, C., Chen, W.: Effective steganalysis based on statistical moments of wavelet characteristic function. In: IEEE International Conference on Information Technology: Coding and Computing, ITCC 2005, April 2005, pp. 768–773. IEEE Computer Society Press, Los Alamitos (2005)
18. Avcibas, I., Memon, N., Sankur, B.: Steganalysis using image quality metrics. Image Processing, IEEE Transactions on 12(2), 221–229 (2003)
19. Watson, A.B. (ed.): Digital images and human vision. MIT Press, Cambridge (1993)
20. Nill, N.: A visual model weighted cosine transform for image compression and quality assessment. Communications, IEEE Transactions on 33(6), 551–557 (1985)
21. A Library for Support Vector Machines, http://www.csie.ntu.edu.tw/cjlin/libsvm/
22. Donoho, D.L., Johnstone, I.M.: Adapting to unknown smoothness via wavelet shrinkage. Journal of the American Statistical Association 90(432), 1200–1224 (1995)
23. Burrus, C.S., Gopinath, R.A., Guo, H.: Introduction to wavelets and wavelets transforms, a primer. Prentice Hall, Englewood Cliffs (1998)
24. Veltkamp, R., Hagedoorn, M.: State-of-the-art in shape matching. Technical Report UU-CS-1999-27, Utrecht University, the Netherlands (1999)
25. Wave files, http://www.wavsurfer.com/
26. Steghide, http://steghide.sourceforge.net/

Author Index